Oracle Fusion Service

How do I use the UI Events Framework?

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

F95864-07

Copyright © 2024, Oracle and/or its affiliates.

Author: DYETTER

Contents

Get Help i

1 Overview of the UI Events Framework 1

Introduction to the UI Events Framework 1
Architecture Details of the UI Events Framework 1
Set Up and Try UI Events Framework 1
What's a context? 4
Operations and Events 7
Understanding the UI Events Framework life cycle 7
Get Field Value Operation 8
Set Field Value Operation 9
Save Record Operation 10
Field Name Mapping 11
Use the UEF Client Object in the Fusion application window 14

2 Learn UI Events Framework Concepts 17

Load an external application inside the Fusion application 17
Load the UI Events Framework inside the third-party application 17
How to use the UI Events Framework 18
Initialize UI Events Framework 18
Get the Corresponding Context for the Use Cases 19
Get the Corresponding Context for Use Cases 21
Generate request object using requestHelper 41
Call the APIs 43

3 Example Use Cases 53

TabOpen Event 53
TabClose Event 53
TabChange Event 54
ContextOpen Event 55
Context Close Event 55
DataLoad Event 56

Field Value Change Event 57
On After Save Event 57
OnBeforeSave Event 58
InvokeServiceConnection Operation 59
FocusTab Operation 60
CloseTab Operation 61
Pop Operations 61
Get Field Value Operation 74
Set Field Value Operation 75
Save Record Operation 76
Publish Get Fields In UI Operation 77
Field Name Mapping 77
Field Value Change Event 80
Use the UEF Client Object in the Fusion application window 81
Extend the Fusion Application Side Pane 82
Modal Window Operations and Events 96
Agent Info Operations 105
Example function with TabContext with given Tab Id 112
Example Function with TabContext 112
Agent Info Operation in Tab Context 113
Pop Operation in TabContext 113
Pop Operation in Current Browser Tab Context 123
getCurrentBrowserTabContext 125
getDependentTabs 125
Subscribe Custom Events 127
Publish Custom Events to all Browser Tabs 129
Communication between External in VB through UEF 132
Custom Event Subscription and Publish from VB 134
Support for Virtual Fields 136
Notifications 140
UI Events Framework and MCA Events 150
UI Events Framework and MCA Operations 155
InteractionLogging 164
Insights 166
CTI Agent Assist Features 174
SetToolbarProperties 180
SetChannelAvailability 181

4 Modal Window Operations and Events 183

Modal Window Operations and Events 183

5 UI Events Framework Reference 185

IUIEventsFramework 185
IUIEventsFrameworkProvider 185
IGlobalContext 193
ITabContext 198
IRecordContext 204
ISidePaneContext 207
IModalWindowContext 212
INotificationContext 213
IMultiChannelAdaptorContext 217
IPhoneContext 218
IEngagementContext 225
ISubscriptionContext 229
iRequestHelper 230
IEventRequest 231
IOperationRequest 232
IFieldValueChangeEventRequest 233
ISetFieldValueOperationRequest 235
ISetFieldValueRequest 235
IGetFieldValueOperationRequest 236
IServiceConnectionRequest 237
IPopFlowRequest 240
IPopFlowInAppRequest 241
IPopFlowAppUIRequest 245
IPopFlowGenericRequest 247
IPopFlowUrlRequest 248
IPopFlowResponse 248
IObjectContext 249
IExtensionResponse 252
IEventResponse 253
IOperationResponse 254
IErrorData 255
IFieldValueChangeData 257
IFieldValueCollection 259

IFieldData 260
ITabEventResponse 262
ITabCloseEventResponse 263
ITabInfo 264
IEngagementInfo 265
ITabChangeEventResponse 267
ITabChangeResponse 268
IFocusTabResponseData 269
ITabCloseOperationResponse 270
IContextOpenEventResponse 272
IContextResponse 273
IGetAgentInfoResponse 274
IGetFieldsInUIResponse 277
IGetFieldsInUIResponseData 278
IGetFieldValueResponse 278
IServiceConnectionResponse 280
IOperationSuccessData 281
ISetFieldValueResponse 281
IOnSaveExtensionContext 282
IOnAfterSaveEventResponse 283
IOnBeforeSaveEventResponse 284
IFieldValueChangeEventResponse 286
ISaveRecordResponse 288
IUpdateSidePaneRequest 289
ISidePaneOpenEventResponse 292
ISidePaneData 293
ISidePaneCloseEventResponse 293
ISidePaneCloseData 294
IOpenModalWindowRequest 295
ICloseModalWindowRequest 297
IOpenPopupWindowRequest 297
ICustomEventSubscriptionRequest 299
ICustomEventSubscriptionResponse 299
ICustomEventRequest 299
ICustomEventResponse 300
ICustomEventResponseData 300
IDataLoadEventResponse 301

IDataLoadEventData 301
INotificationActionPayload 302
IShowNotificationRequest 302
INotificationCloseActionData 304
INotificationActionData 304
IMcaActionRequest 305
IMcaNewCommEventActionRequest 305
IMcaStartCommEventActionRequest 306
IMcaCloseComActionData 306
IMcaCloseCommEventActionRequest 307
IMcaFeedLiveTranscriptActionRequest 307
IMcaFeedSystemMessageActionRequest 308
IMcaNewComActionResponse 310
IMcaStartComActionResponse 310
IMcaCloseComActionResponse 310
IMcaNewComActionData 311
IMcaStartComActionData 311
IMcaEngagementData 312
IMcaOutData 313
IMcaStartCommEventOutData 313
IMcaCloseCommEventOutData 314
IMcaInData 314
IMcaStartCommInData 315
IMcaCloseCommInData 315
IMcaInDataRequest 316
IMcaStartCommInDataRequest 319
IMcaCloseCommInDataRequest 324
IMcaComEventActionData 330
IMcaNewComEventActionData 330
IMcaStartComEventActionData 331
IMcaCloseComEventActionData 331
IMcaOutDataResponse 332
IMcaCloseCommEventOutDataResponse 336
IMcaStartCommEventOutDataResponse 340
IMcaGetConfigurationActionRequest 345
IMcaAgentStateEventActionRequest 345
IMcaDisableFeatureActionRequest 346

IMcaReadyForOperationActionRequest 347
IMcaGetConfigurationActionResponse 347
IMcaDisableFeatureActionResponse 347
IMcaReadyForOperationActionResponse 348
IMcaAgentStateEventActionResponse 348
IMcaOutboundCommErrorActionRequest 348
IMcaOutBoundCommErrorActionResponse 349
IMcaGetConfigurationActionResponseData 349
IMcaOutBoundCommErrorActionResponseData 349
IMcaAgentStateEventActionResponseData 350
IMcaDisableFeatureActionResponseData 351
IMcaReadyForOperationActionResponseData 351
IMCAGetConfigurationResponsePayload 351
IMCAOutBoundCommErrorResponsePayload 352
IMCAAgentStateEventResponsePayload 353
IMCAConfiguration 354
IMCAGetConfiguration 355
IMcaEventRequest 356
IMcaOnDataUpdatedEventResponse 356
IMcaOnDataUpdatedData 356
IMcaOnDataUpdatedOutDataResponse 357
IMcaOnDataUpdatedOutData 357
IMcaOnDataUpdatedEventData 358
IMcaEventData 358
IMcaonOutgoingEventResponse 358
IMcaOnOutgoingEventData 359
IMcaOnOutgoingEventOutData 359
IMcaOnOutgoingEventOutDataResponse 360
IMcaOnToolbarInteractionCommandEventResponse 361
IMcaOnToolbarAgentCommandEventResponse 361
IMcaOnToolbarInteractionCommandDataResponse 362
IMcaOnToolbarAgentCommandDataResponse 363
IMcaOnToolbarInteractionCommandData 364
IMcaOnToolbarAgentCommandDataResponse 365
IInteractionLogger 366
IInsightsContext 369
IInsightsSubscriptionRequest 372

IShowInsightsRequest 372
IInsightsActionRequest 373
IDismissInsights 374
IInsightsActionEventResponse 374
IInsightsActionData 374
IInsightsDismissActionEventResponse 375
IInsightsDismissActionData 375
IInsightsOperationResponse 376
IInsightsOperationResponseData 376
IInsightsGetAllOperationResponse 376
IInsightsGetAllOperationResponseData 377
IToolbarContext 377
ISetToolbarPropertiesActionRequest 378
ISetToolbarPropertiesActionResponse 380
ISetToolbarPropertiesActionResponseData 381
ISetChannelAvailabilityRequest 382
ISetChannelAvailabilityResponse 384

6 MCA integration 387
Overview of integrating MCA with different suppliers 387
CTI scenarios 390

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

Some application pages have help icons to give you access to contextual help. 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. If the page has contextual help, help icons will appear.

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 Overview of the UI Events Framework

Introduction to the UI Events Framework

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

The framework has dedicated APIs to listen to events generated from the the Fusion application and perform operations inside the Fusion application. The UI Events Framework manages these events and operations. For example, an input Text Field in the Fusion application 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 the Fusion application, 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.

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 Fusion 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 the Fusion application.
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 Fusion 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 the Fusion application

You use the `<oj-cx-svc-common-ui-events-container>` component to load external applications in the Fusion application. You install the component from Oracle Component Exchange and configure it in the Fusion application 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 Fusion application 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

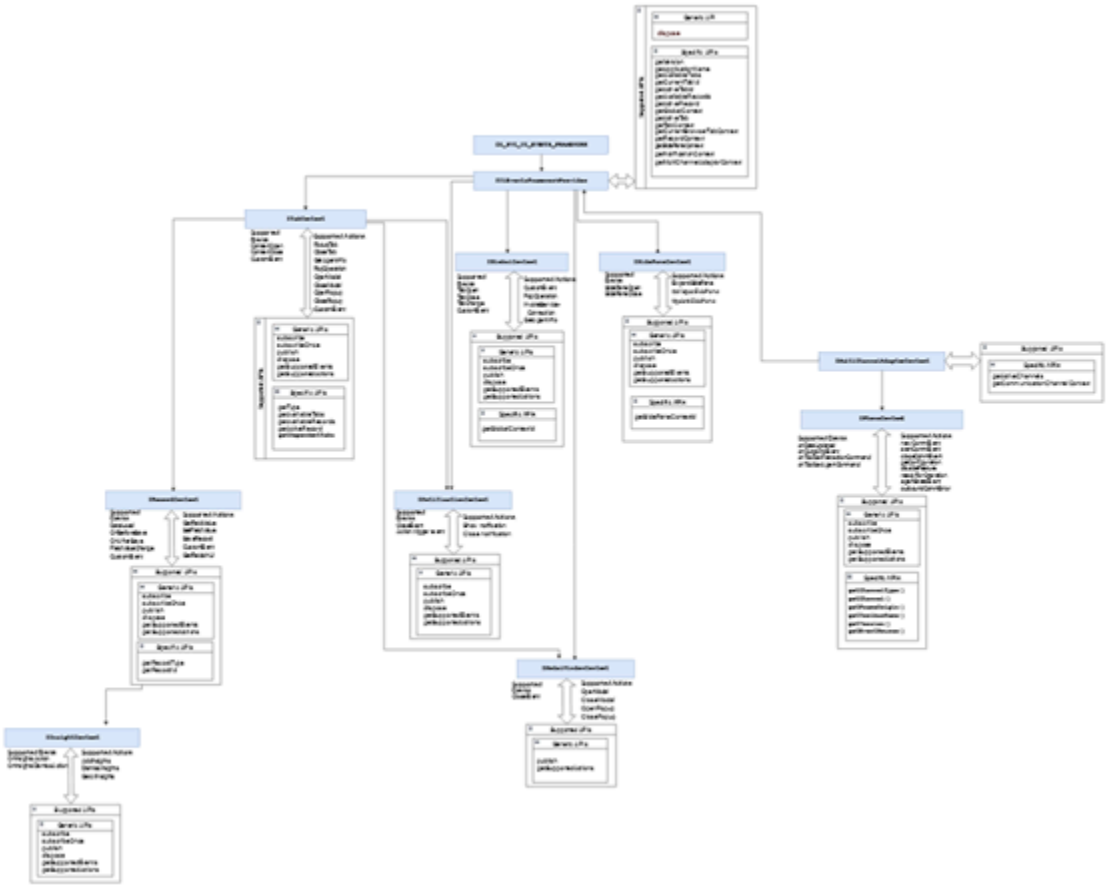
Update the request type and subscription in the above code as given in the following example. Then, transpile the code and reload the iframe.

```
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);
 });
}
```

What's a context?

Context is an entity which encapsulates the operations and events related a specific area.

For example, all the APIs related to an object are encapsulated in the recordContext. Once the framework is initialized and the context is obtained, you can perform record level operations such as set an input field value of a record, get a value loaded into a text area, or if the agent updates the value of a field, an event can be fired on the record level. There are other contexts for specific use cases.



Currently available contexts are shown in the following list:

- *GlobalContext*: This context encapsulates the events and operations related on a global level. An example would be, opening an SR in a new MSI tab or browser tab is an event which is happening on a global level. So a tab open event is something that you can use to listen to on a GlobalContext. The type details are available in *IGlobalContext*.
- *TabContext*: At this context level you can focus or close a specific tab. Or you can subscribe to an event context open, meaning the content loaded into the specific tab can be accessed when this event is fired. The type details are available in *ITabContext*.
- *Record Context:*: Encapsulates field related operations an events. For instance, setting a field value, read a value from an input field, or listening to a field value change. The type details are available in*IRecordContext*
- *SidePaneContext*: With this context, we can expand or collapse a specific side pane, we can update the content of a side pane. We can subscribe to side pane open and close events. The type details are available in *ISidePaneContext*
- *Modal Window Context:* : At this context level, we can open/close Modal/Popup(s).
- *NotificationContext*: At this context level, we can open/close Show and close notifications and listen to the close action and action trigger event of the notifications.
- *MultiChannelAdaptorContext*: This object encapsulates the events and actions on different channels when there's a telephonic or chat interaction happening.
- *PhoneContext*: This object can be derived from MultiChannelAdaptorContext object by calling `getCommunicationChannelContext` by passing channel type as `PHONE`.

- *InsightsContext*: At this context level, we can open/dismiss insights and listen to the close action and action trigger event of the insights.

Think of it like this: a TabOpen event is triggered at GlobalContext level. The RecordOpen event is triggered at the Tab Context level, and a FieldValueChange event is triggered at a RecordContext level.

Here's more details:

Context Type	Supported Events	Supported Operations	API to get the context
GlobalContext	TabOpen, TabClose, TabChange, CustomEvent	PopOperation (open a tab/record pop a page), InvokeServiceConnection, GetAgentInfo, CustomEvent	getGlobalContext(): Promise<IGlobalContext>;
TabContext	ContextOpen, ContextClose, CustomEvent	FocusTab, CloseTab, PopOperation, GetAgentInfo, CustomEvent, OpenModal, CloseModal, OpenPopup, ClosePopup	getTabContext(browserTabId?: string): Promise<ITabContext>;
RecordContext	FieldValueChange, OnBeforeSave(Controllable Event), OnAfterSave, OnDataLoad, CustomEvent	GetFieldValue, SaveRecord, SetFieldValue, CustomEvent	getAvailableRecords(tabId?: string): Promise<IRecordContext[]>;
SidePaneContext	SidePaneOpen, SidePaneClose	Expand Side pane, Collapse Side pane, Update Side pane	getSidePaneContext(sidePaneId: string): Promise<ISidePaneContext>;
ModalWindowContext	NA	Open Modal, Close Modal, Open Popup, Close Popup	getModalWindowContext(): Promise<IModalWindowContext>;
MultiChannelAdaptorContext	NA	NA	getMultiChannelAdaptorContext(): Promise<IMultiChannelAdaptorContext>;
PhoneContext	OnToolbarAgentCommand, OnToolbarInteractionCommand, OnDataUpdated, OnOutgoingEvent	NewCommEvent, StartCommEvent, CloseCommEvent, OutboundCommError, GetConfiguration, DisableFeature, ReadyForOperation, AgentStateEvent	getCommunicationChannelContext(channelTy McaChannels): Promise<ICommunicationChannelContext>;
NotificationContext	OnNotificationAction, OnNotificationCloseAction	ShowNotification, CloseNotification	getNotificationContext(): Promise<INotificationContext>;
InsightsContext	OnInsightsAction, OnInsightsDismissAction	ShowInsights, DismissInsights	getInsightsContext(): Promise<IInsightsContext>;

Note: OnBeforeSave (in the Record Context) is fired before firing the REST API that commits the data into the server. You can do asynchronous operations in this callback or you can cancel this event. This event can wait for an asynchronous operation to be completed

Operations and Events

Once a context is ready you can perform various operations or subscribe to various events available in that specific context.

Set Field Value Operation Example

```
const requestObject: ISetFieldValueOperationRequest = frameworkProvider
 .requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as ISetFieldValueOperationRequest;
requestObject.field().setValue(key, value);
recordContext.publish(requestObject).then((response: IOperationResponse) => {
 console.log((response as ISetFieldValueResponse).getResponseData().getMessage());
}).catch((error) => console.log(error));
```

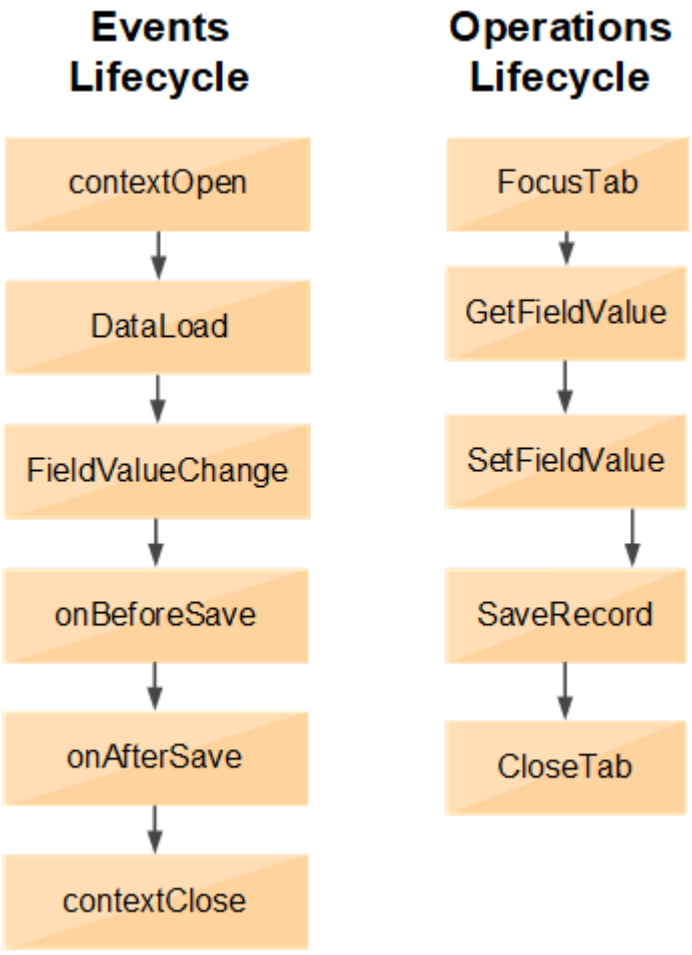
Subscribe to an Event Example

```
const requestObject: IFieldValueChangeEventRequest = frameworkProvider
 .requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent') as
 IFieldValueChangeEventRequest;
requestObject.setFields(['ServiceRequest.Title']);
let subscription = recordContext.subscribe(requestObject, (response: IEventResponse) => {
 const fieldName = (response as any).getResponseData().getFieldName();
 const newValue = (response as any).getResponseData().getNewValue();
 const oldValue = (response as any).getResponseData().getOldValue();
 console.log('FieldValueChangeEvent', fieldName, 'oldValue:', oldValue, 'newValue:', newValue)
});
```

Understanding the UI Events Framework life cycle

Here's an overview of the Events life cycle.

The following graphic presents a life cycle overview:



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','V1');
 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', 'v1');
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', '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');
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', 'v1');
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', 'v1');
 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', 'v1');
 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
```

```
{
  "Resources": {
    "serviceRequests": {
      "discrColumnType": false,
      "title": "Service Request",
      "titlePlural": "Service Requests",
      "group": "OEC",
      "usage": "BusinessObject",
      "attributes": [
        {
          "name": "Title",
          "type": "string",
          "updatable": true,
          "mandatory": true,
          "inputRequired": true,
          "queryable": true,
          "allowChanges": "always",
          "precision": 400,
          "title": "Title",
          "maxLength": "400",
          "properties": {
            "DISPLAYWIDTH": "40",
            "FIELDORDER": "0.0"
          },
          "annotations": {
            "description": "The brief title of the service request."
          }
        },
        {
          "name": "SeverityCd",
          "type": "string",
          "updatable": true,
          "mandatory": false
```

Here the metadata of 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
eg: ServiceRequest.Title
```

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.

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 must call the Publish API on top of record context ito 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', 'v1');
 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', 'v1');
 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 must call the publish API on top of record context 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', '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');
 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.

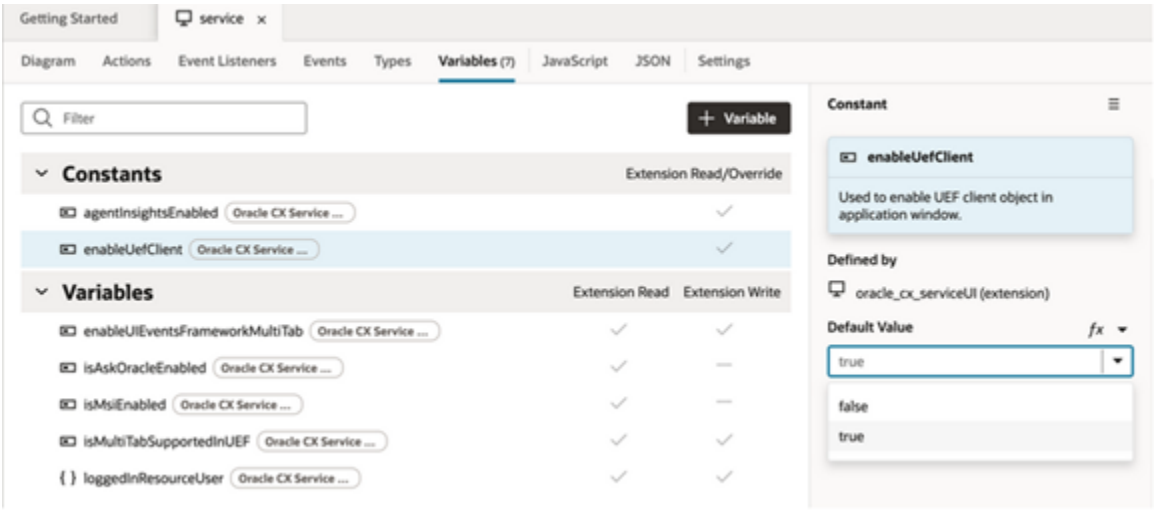
```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 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
 });
```

Use the UEF Client Object in the Fusion application window

enableUefClient is an application level variable that can be exposed in the Fusion application.

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.



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 the Fusion application to access it. This feature coexists with UEF objects consumed inside external applications.

Here's a JavaScript sample added in the Fusion application 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}`);
 });
}
```

2 Learn UI Events Framework Concepts

Load an external application inside the Fusion application

You use the `<oj-cx-svc-common-ui-events-container>` component to load external applications in the Fusion application.

You install the component from Oracle Component Exchange and configure it in the Fusion application 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>
```

Here are the steps:

1. Install the custom component <oj-cx-svc-common-ui-events-container> from Oracle Component Exchange.
2. Add the new custom component to the area you want in the Fusion application.
3. Provide your external application URL in the URL property of the <oj-cx-svc-common-ui-events-container> component.

Load the UI Events Framework inside the third-party application

For an external application to interact with the Fusion application, the application must have the UI Events Framework library file added.

The library file is available at *https://static.oracle.com/cdn/ui-events-framework/libs/ui-events-framework-client.js*.

To add the UI Events Framework library file to the external application, you can use any native method of loading scripts into an HTML file.such as loading the library file in an HTML using the script tag like this example:

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

Type declarations are available in uiEventsFramework.d.ts. Add this file to the TypeScript project for type checking. Once the library is loaded in the external application, following object will be available in the application's DOM.

```
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework
```

You can use APIs exposed over this object to interact with the Fusion application and other Fusion applications.

How to use the UI Events Framework

To use any API exposed in the UI Events Framework, you must follow these steps:

Prerequisites

1. Load the external application in a Fusion application using the `<oj-cx-svc-common-ui-events-container>` component.
2. Load the UI Events Framework library file in external application's HTML or DOM file.

Required Steps

1. *Initialize UI Events Framework*
2. *Get the corresponding context for the use cases*
3. *Generate the request object using requestHelper*
4. Call the APIs exposed in the previous contexts. The APIs are:
    - *Subscribe API*
    - *SubscribeOnce API*
    - *Publish API*
    - *Dispose API*

Initialize UI Events Framework

You must initialize the framework using an application name for any interaction. Once the framework is initialized successfully, you can get the reference to the instance of `IUiEventsFrameworkProvider`. All the interactions occurring with this object can be tracked with the application name you provide during the initialization.

Here's the syntax:

```
initialize(applicationName: string, version: string): Promise<IUiEventsFrameworkProvider>;
```

Here are the parameters:

Parameters

Parameter Name	Required?	Description
applicationId	Yes	You must call the `initialise` function with a unique application ID to use the UI Events Framework. The application ID helps in debugging during testing.

Parameter Name	Required?	Description
version	No	Oracle aims to retain backward compatibility for all releases. If in the future, the framework deprecates some APIs without backward compatibility, you can use this parameter to load older or newer versions of the library. Currently, Version will have the default value as v1. As of this time, you can't set this version value.

Here's an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','v1');
```

Here's an example in Javascript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','v1');
```

Get the Corresponding Context for the Use Cases

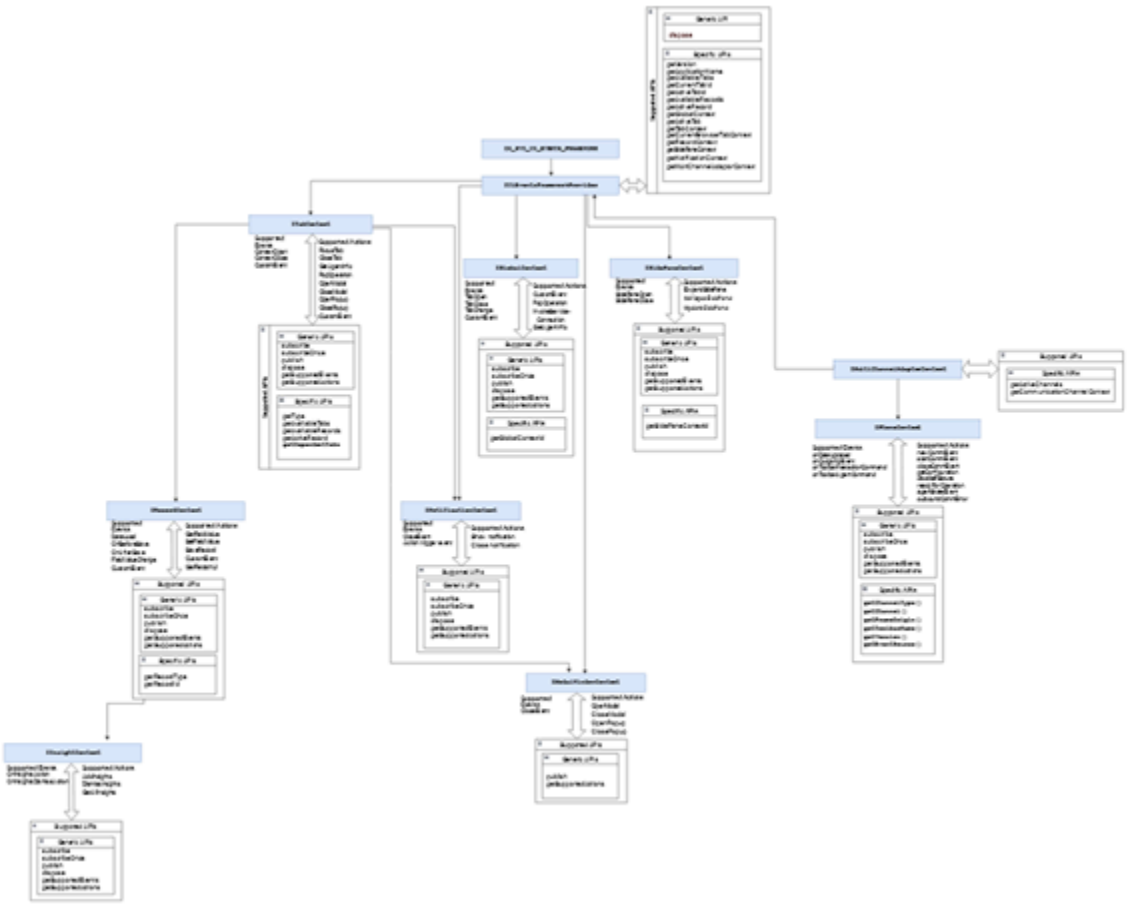
Once you get the instance of the `IUiEventsFrameworkProvider` you can now choose the context based on your use cases. The `UiEventsFrameworkProvider` has seven contexts which are shown in the following table:

Contexts

Context	Description	Topic
GlobalContext	Enables you to listen to application-level events and perform application-level operations.	*IGlobalContext*
TabContext	Enables you to listen to tab level events.	*ITabContext*
RecordContext	Enables you to listen to all those events or operations related to an object.	*IRecordContext*
SidePaneContext	Enables you to access sidePane's events and perform actions onsidePane in the Fusion application.	*ISidePaneContext*
ModalWindowContext	Enables you to perform actions on the ModalWindow in the Fusion application.	*IModalWindowContext*
NotificationContext	Enables you to perform Notification actions and events in the Fusion application.	*INotificationContext*

Context	Description	Topic
InsightsContext	Enables you to perform Insights CCA related actions and events from a particular record context	*InsightsContext*

All the extensible events and actions of the UI Events Framework are supported on top of contexts like GlobalContext, TabContext and RecordContext. The following diagram shows the hierarchy of these contexts maintained in the framework along with the supported events, actions and APIs supported in each of the contexts. The sections which follow explain each context in detail.



Get the Corresponding Context for Use Cases

GlobalContext

GlobalContext enables you to listen to application-level events and perform application-level operations. It enables listening to tabOpen, tabClose and tabChange events. It also supports TabFocus, TabClose, CustomEvent and InvokeRestApi actions.

To get the reference of GlobalContext, you can use the getGlobalContext function. This function provides a reference to the application GlobalContext.

Syntax

Here's the syntax:

```
getGlobalContext(): Promise<IGlobalContext>;
```

Code Samples

Here's code sample in TypeScript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

Here's code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
```

Supported Events

Supported Events

Event Name	Description	Example
TabOpen	Started when a new browser tab or an MSI tab is opened.	*TabOpen Event*
TabClose	Started when a new browser tab or an MSI tab is changed.	*TabClose Event*
TabChange	Started when a browser tab or MSI tab switch happens.	*TabChange Event*
CustomEvent	Used to support any business specific custom events.	*Subscribe Custom Events*

Supported Operations

Supported Operations

Action Name	Description	Example
PopOperation	Used to open any page like an in app page or a custom page in a new browser tab or a new MSI tab.	*Pop Operation*
InvokeServiceConnection	Used to call any REST API in VB and to fetch its response in an external application.	*InvokeServiceConnection Operation*
GetAgentInfo	Used to fetch information about current signed in user.	*Agent Info Operation in Global Context*
CustomEvent	Used to support any business specific custom events.	*Publish Custom Events*
setChannelAvailability	Enables or disables the phone icon	*SetChannelAvailability*

Supported Methods

Supported Operations

Method Name	Description
subscribe	Used to subscribe to an event fired from the the Fusion application.
subscribeOnce	Used to subscribe to the Fusion application event just once.
publish	Used to perform certain operations.
dispose	Used to dispose of, or unregister any subscriptions that were added to an event.
getSupportedEvents	Used to get the list of supported events in GlobalContext.
getSupportedActions	Used to get the list of supported operations in GlobalContext.
getGlobalContextId	Used to get the unique context ID of the GlobalContext object.

TabContext

TabContext enables you to listen to tab level events such as ContextOpen, ContextClose and perform tab level operations such as FocusTab and CloseTab.

To get the reference of TabContext, users can use the getTabContext function. This function provides a reference to the active tab in which the application is loaded. They can pass the browserTabId as an optional parameter to get a particular browser tab's context.

Syntax

Here's the syntax:

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

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

getActiveTab(): Promise<ITabContext>;
```

Parameters

Parameter Name	Required?	Description
browserTabId	No	A particular browser tab's ID. **Note:** You can get each browser's tabId value by calling getCurrentTabId API on frameworkProvider from each browser tab.

Code Samples

Here's code sample in TypeScript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();

/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();

/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getActiveTab();
```

Here's code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();

const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

```
const tabContext = await frameworkProvider.getTabContext();


const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getActiveTab();
```

Supported Events

Supported Events

Event Name	Description	Example
ContextOpen	Started when a new object gets opened in a particular tab.	*ContextOpen Event*
ContextClose	Started when an object gets closed in a particular tab.	*Context Close Event*
CustomEvent	Used to support any business specific custom events.	*Subscribe Custom Events*

Supported Operations

Supported Operations

Action Name	Description
FocusTab	Used to focus a particular browser or MSI tab.
CloseTab	Used to close a particular browser or MSI tab.
PopOperation	Used to to pop any page such as in an app page or a custom page in the specified tab context.
GetAgentInfo	Used to fetch the information about current logged in user.
CustomEvent	Used to support any business specific custom events.
OpenModal	Used to open a modal window.
CloseModal	Used to close a modal window.
OpenPopup	Used to open a dialog box.
ClosePopup	Used to close a dialog box.

Supported Methods

Supported Operations

Method Name	Description
subscribe	Used to subscribe to an event fired from the Fusion application.
subscribeOnce	Used to subscribe to a the Fusion application event just once.
publish	Used to perform certain operations.
dispose	Used to dispose of, or unregister any subscriptions that were added to an event.
getSupportedEvents	Used to get the list of supported events in *GlobalContext*.
getSupportedActions	Used to get the list of supported operations in*GlobalContext*.
getType	Returns the type of a *tabContext* object.
getAvailableTabs	Returns all the available tabs on top of a specific tab or its *tabContext*.
getAvailableRecords	Returns all the available records on a specific tab.
getActiveRecord	Returns the active record on a specific tab.
getDependentTabs	Returns all the child tabs opened from the tab context.

RecordContext

RecordContext enables you to listen to events and operations related to an object. For instance, you can listen to a field value change, on before save event, ON after save event, and so on.

Supported Events

Supported Events

Event Name	Description	Example
FieldValueChange	Started when a field value change, such as an SR title field change, occurs.	*Field Value Change Event*
OnBeforeSave	Started before start of the REST API request to commit the data to the server. The end-user can perform asynchronous operations in this callback or cancel this event. This event	*OnBeforeSave Event*

Event Name	Description	Example
	can wait for an asynchronous operation to be completed.	
OnAfterSave	Started after a record is saved a in the VB application. For example, after saving an SR.	*On After Save Event*
OnDataLoad	Started when data is loaded. For example, after fetching an SR object data.	*DataLoad Event*
CustomEvent	Used to support any business specific custom events.	*Subscribe Custom Events*

Supported Operations

Supported Operations

Operation Name	Description	Example
GetFieldValue	Used to fetch the current value of a field. For instance, us it to get the current value of the Title field of an SR.	Publish *GetFieldValue operation*.
Save Record	Used to save an open record. For instance, use it to save an SR form from an edit page or save an SR from create form.	Publish *Save Record operation*.
SetFieldValue	Used to update the value of a particular field. For instance, use it to update the Title field of an SR.	Publish *SetFieldValue operation*.
CustomEvent	Used to support any business specific custom events.	*Publish CustomEvent operation*.
GetFieldsInUI	Used to fetch the list of fields in the rendered dynamic form.	*Publish GetFieldsInUI operation*.

Supported Methods

Supported Methods

Method Name	Description
subscribe	Subscribe to an event from the Fusion application. For example, the *Field Value Change Event*
subscribeOnce	Subscribe once to an event from fusion application's record context, once the event is fired, the subscription disposes. For example: *Field Value Change Event*.

Method Name	Description
publish	Publish an operation in record context, for example: *SetFieldValue operation*, *GetFieldValue operation*.
dispose	Disposes the *recordContext*.
getSupportedEvents	Returns events supported by that record context as an array, for example: *GetFieldValue operation*, *SetFieldValue operation*, *Save Record operation*, *Subscribe Custom Events*.
getSupportedActions	returns actions supported by that record context as an array, for example: *DataLoad Event*, *OnBeforeSave Event*, *On After Save Event*, *Field Value Change Event*, *Subscribe Custom Events*.
getRecordType	Returns a type of record, for example: ServiceRequest, or Case.
getRecordId	Returns the record ID. For a record context of a record not created yet, the ID will be a negative number.

You can get the reference of RecordContext only by calling getActiveRecord() API on top of TabContext object which could be either a browser-TabContext or an MSI-TabContext. Apart from getActiveRecord API, you can get the RecordContext object as a response of ContextOpen event subscription.

SidePaneContext

The object that encapsulates the UEF-provided sidePane's events and actions support. This object is added on top of UEF provider object to access sidePane's events and perform actions on sidePane actions in the Fusion application. You can get the sidePaneContext by calling the getSidePaneContext API provided in the UEF provider object.

Syntax

Here's the syntax:

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

The following code Typescript sample shows how you can pass sidePanelId to access SidePaneContext:

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

The following code JavaScript sample shows how you can pass sidePanelId to access SidePaneContext:

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

> **Note:** See the Configuration of ServiceCenter/fusionapp SidePane section to know how user can configure UEF Supported advanced side pane in fusion application. The sidePaneContext refers to only the advanced sidePane, the UEF legacy sidePane can't be controlled using this object.

Supported Operations

Supported Operations

Operation Name	Description	Example
ExpandSidePane	Used to expand a particular SidePane.	ExpandSidePane
CollapseSidePane	Used to collapse a particular SidePane.	CollapseSidePane
UpdateSidePane - setVisiblity	Used to set the visibility (true or false) of a particular SidePane.	UpdateSidePane - setVisiblity
UpdateSidePane - setIcon	Used to set icon for a particular SidePane Icon.	UpdateSidePane - setIcon
UpdateSidePane - setSectionId	Used to set section ID for a particular SidePane.	UpdateSidePane - setSectionId

Supported Events

Supported Events

Event Name	Description
SidePaneOpen	Started when a SidePane is opened.
SidePaneClose	Started when a SidePane is closed.

Note: These events notifications are started for the particular sidePaneContext object corresponding to a sidePaneId on which user has added a subscription.

Supported Methods

subscribe	Function to listen to a SidePaneContext supported event, such as SidePaneOpen or SidePaneClose.	subscribe
subscribeOnce	Function to listen once a SidePaneContext supported event such as SidePaneOpen, or SidePaneClosed is launched.	subscribeOnce
publish	Function to publish a SidePane supported operation on SidePaneContext object. For example, ExpandSidePane, CollapseSidePane, UpdateSidePane	publish

dispose	Function to dispose/remove all events subscribed on SidePaneContext object.	dispose
getSupportedEvents	Function to get all the supported events on SidePaneContext object.	getSupportedEvents
getSupportedActions	Function to get all the supported actions on SidePaneContext object.	getSupportedActions
getSidePaneContextId	Function to get the sidePane contextId of the SidePaneContext object.	getSidePaneContextId

ModalWindowContext

ModalWindowContext is a context in the UEF provider object used to perform actions in modal and dialog box.

You can get the ModalWindowContext by calling getModalWindowContext API provided in UEF provider object.

Syntax

Here's the syntax:

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

Here's how you can access ModalWindowContext:

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

Supported Operations

Supported Operations

Operation Name	Description	Example
OpenModal	Used to open the model window.	OpenModal operation
CloseModal	Used to close the modal window.	CloseModal operation
OpenPopup	Used to open a dialog box.	OpenPopup operation
ClosePopup	Used to close a dialog box.	ClosePopup operation

Supported Methods

Supported Methods

Method Name	Description	Example
publish	Publishes an operation in record context. For instance, Open Modal, Close Modal, Open Popup, Close Popup	Publish
getSupportedActions	Returns actions supported by that record context as an array.	Get supported actions

Note: All the modal window operations work with GlobalContext and TabContext.

MultiChannelAdaptorContext

Fusion Application has legacy Multi Channel Adaptor APIs exposed that third party applications can use to enable real time communications over phone, chat, and so on. UEF supports all the legacy MCA APIs through UEF and this approach has some advantages over the legacy integration as given shown in this topic.

Advantages of using MCA through UEF are as follows:

- Control and Access VB objects from MCA actions.
- Single client library needed for third party integration. (injection of mcaInteractionV1.js can be avoided which is used for integration of legacy MCA).
- Typescript support for MCA events and actions.
- Single coding pattern experience for all third party integration, such as both UEF and MCA.
- Perform multiple operations from third party client app on MCA action completions, such as Pop more screens

  Note: This implementation is backward compatible and shouldn't affect the existing integrations with MCA.

MultiChannelAdaptorContext is the object that encapsulates the events and actions on different channels when there's a telephonic or chat interaction happening. This object is added on top of UEF provider object to access a CommunicationChannelContext, for example, PhoneContext. Users can get the MultiChannelAdaptorContext by calling getMultiChannelAdaptorContext API provided in UEF provider object.

Syntax

Here's the syntax:

```
getMultiChannelAdaptorContext(): Promise<IMultiChannelAdaptorContext>;
```

Here's a Typescript example of how to access multiChannelAdaptorContext.

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

Here's a Javascript example of how to access multiChannelAdaptorContext:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

Supported Methods

Supported Methods

Method Name	Description	Example
getActiveChannels	Gets supported channels, such as Phone, Chat, and so on.	See getActiveChannels in*IMultiChannelAdaptorContext*
getCommunicationChannelContext	Gets the context of the communication channel supported over multiChannelAdaptorContext object, such as PhoneContext.	See getCommunicationsChannelContext in *IMultiChannelAdaptorContext*

PhoneContext

Encapsulates the UEF provided telephonic events and actions. It can be derived from MultiChannelAdaptorContext object by calling the getCommunicationChannelContext API by passing channelType as PHONE.

Syntax

Here's the syntax:

```
getCommunicationChannelContext(channelType: McaChannels): Promise<ICommunicationChannelContext>;
```

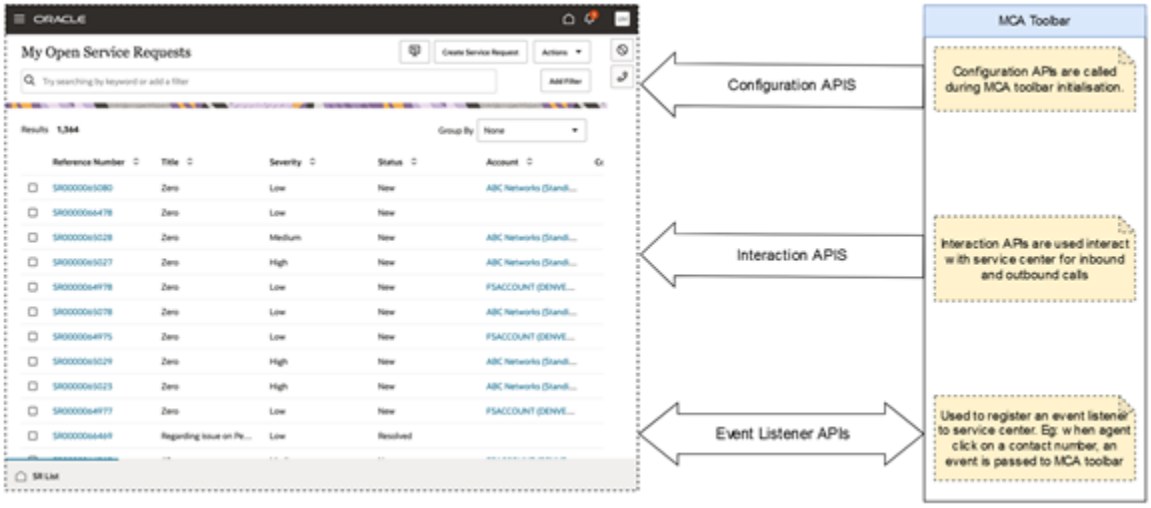
Here's a Typescript example of how you can access PhoneContext by calling the getCommunicationChannelContext:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
```

Here's a Javascript example of how you can access PhoneContext by calling the getCommunicationChannelContext:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
```

Here's a diagram that shows the events and actions supported by PhoneContext.



Supported Events

Supported Events

Event Name	Description	Example
OnToolbarAgentCommand	This is a controllable event. This means, to control this event by either resolving or rejecting it, you must pass a promise as the return of this event callback function. When notified of this event subscription, you can check the command, getActiveInteractionCommands, in the event response and pass outData for that command by resolving the promise with that outData. The event subscription can be added only once during initialization of the MCA code. You'll get this event notification only once per session.	*onToolbarAgentCommand*
OnToolbarInteractionCommand	This is a controllable event. To control this event by either resolving or rejecting it, you must pass a promise as the return of this event callback function.When notified of this event subscription, you can check the command in the event response. Commands supported are, accept, disconnect, reject, setActive. These commands correspond to accept, disconnect, and reject operations an agent might make in the Fusion application. You can wire up their logic for each of these commands, on the callback of this event response.	*onToolbarInteractionCommand*

Event Name	Description	Example
onDataUpdated	Used to listen to a DataUpdate event that happens in the Fusion application. It's a non controllable event	*onDataUpdated*
onOutgoingEvent	Used to listen to an outBound call event that happens in the Fusion application. It's a non controllable event. You can wire the logic for the outbound call connection on notification of this event.	*onOutgoingEvent*

Supported Operations

Supported Operations

Operation Name	Description	Example
newCommEvent	This is the action user needs to publish when there's a ring received scenario. For example, when a user is trying to call an agent, Fusion application should show a notification with Accept or Decline button. To enable this the MCA integrated CTI should publish a newCommEvent request.	*newCommEvent*
startCommEvent	This is the action user needs to publish when there's a call start scenario if the newCommEvent. For example, for an Inbound call scenario, user should listen to onToolbarInteractionCommand and when the user gets a notification of it with the command as Accept, they'd then need to publish this action to establish that call connection.	*startCommEvent*
closeCommEvent	This is the action user needs to publish when there's a call decline scenario. For example, when an agent clicks Decline on a call notification, the CTI integration code base gets a notification for onAgentInteractionCommand event with the command as Decline, if it has a subscription added for the same.	*closeCommEvent*
outboundCommError	The toolbar can publish this API to notify the Fusion application that an error occurred during initiation of the outbound event. The error occurs if the identifier of the event, such as phone number, or email can't be used to establish a connection.	*outboundCommError*

Operation Name	Description	Example
getConfiguration	To get configuration information that enables the toolbar to evaluate the features supported by the Fusion application	*getConfiguration*
disableFeature	Informs the Fusion application that a subset of available functionality must be disabled because the toolbar hasn't implemented it.	*disableFeature*
readyForOperation	Notifies the Fusion application that the toolbar is ready for operation	*readyForOperation*
agentStateEvent	Notifies the Fusion application of a change in the user's signed in or availability status for the specified channel	*agentStateEvent*

Supported Methods

Supported Methods

Method Name	Description	Reference
subscribe	Function to listen to a PhoneContext supported Event. For example, onToolbarInteractionCommand, onDataUpdated, onOutgoingEvent.	*Subscribe API*
subscribeOnce	Function to listen only once a PhoneContext supported Event. For example, onToolbarInteractionCommand, onDataUpdated, onOutgoingEvent.	*SubscribeOnce API*
publish	Function to publish a PhoneContext supported operation on SidePaneContext object. For example, getConfiguration, agentStateEvent, readyForOperation, disableFeature, newCommEvent, startCommEvent, closeCommEvent, outboundCommError.	*Publish API*
dispose	Function to dispose or remove all events subscribed on PhoneContext object.	*Dispose API*
getSupportedEvents	Function to get all the supported events on PhoneContext object.	See getSupportedEvents entry in *IPhoneContext*..
getSupportedActions	Function to get all the supported actions on PhoneContext object.	See getSupportedActions entry in *IPhoneContext*..

Method Name	Description	Reference
getChannelType	Function to get the channel type in PhoneContext API requests, such as, ORA_SVC_PHONE.	See getChannelType entry in *IPhoneContext*..
getChannel	Function to get the type in context object. The value is PHONE.	See getChannel entry in *IPhoneContext*..
getFrameOrigin	Function to get the frame origin in PhoneContext object requests.	See getFrameOrigin entry in *IPhoneContext*..
getToolbarName	Function to get the toolbar name in PhoneContext object requests.	See getToolbarName entry in *IPhoneContext*..
getVersion	Function to get the toolbar name in PhoneContext object requests.	See getVersion entry in *IPhoneContext*..
getEventSource	Function to get the event source in PhoneContext object requests.	See getEventSource entry in *IPhoneContext*..

EngagementContext

This context we get in the response of startCommEvent operation's response by calling getEngagementContext API.

This object will encapsulate the tabContext in which the new call connection has been established, which means the user can get control over the objects being opened by that call connection. The getEngagementContext API can be called from IMcaStartComActionData which is the object we get by calling getResponseData API on the IMcaStartComActionResponse object. IMcaStartComActionResponse object is the response of StartCommEvent operation.

Syntax

Here's the syntax:

```
getEngagementContext(): IEngagementContext;
```

Here's a Typescript example of how you can access EngagementContext:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
    // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
        request.setAppClassification('appClassfication');
     request.setEventId('1'); // Set the event ID here
```

```
// request.setInputData(_inboundData);// _inboundData of type - IMcaStartCommInData
const inData: IMcaStartCommInDataRequest = request.getInData();
phoneContext.publish(request).then((res: IOperationResponse) => {
const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
const startCommData: IMcaStartComActionData = response.getResponseData();
const engagementContext: IEngagementContext = startCommData.getEngagementContext();
})
```

Here's a Javascript example of how you can access EngagementContext:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer to Application Classification Code for the list of supported app classifications
request.setAppClassification('appClassfication');
request.setEventId('1'); // Set the event ID here
const inData = request.getInData();
phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
})
```

Supported Operations

Supported Operations

Method Name	Description	Example
FeedLiveTranscript	This operation is used to add live transcript messages during an ongoing phone call.	FeedLiveTranscript
FeedSystemMessage	This operation is used to add system messages during an ongoing phone call.	FeedSystemMessage

Supported Methods

Supported Methods

Method Name	Description	Example
getEngagementId	The unique ID of the engagement object.	See getEngagementId entry in *IEngagementContext*..
getTabContext	The TabContext in which the engagement is opened in the UI. Using this context you can access the records opened in that tabContext and perform actions. All APIs added on a TabContext object will hold true on the return value of this API.	See getTabContext entry in *IEngagementContext*..

Method Name	Description	Example
getEngagementData	The function to fetch the data in that engagement object.	See getEngagementData entry in *IEngagementContext*..
getChannelType	The function to get the channel type in PhoneContext API requests, such as ORA_SVC_PHONE.	See getChannelType entry in *IEngagementContext*..
getChannel	The function to get the type in context object. The value is PHONE.	See getChannel entry in *IEngagementContext*..

NotificationContext

NotificationContext enables you to perform actions and events on Notifications. For example, show Notification, Close Notification, Listen to close notification, Listen to actions in notification. To get the reference to NotificationContext, users can use the getNotificationContext function.

Syntax

Here's the syntax:

```
getNotificationContext(notificationId: string): Promise<INotificationContext>;
```

Here's a Typescript example of the getnotification context from the uefFrameworkProvider:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId');
```

Here's a Javascript example of the getnotification context from the uefFrameworkProvider:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const notificationContext = await frameworkProvider.getNotificationContext('notificationId');
```

NotificationContext also enables you to do actions and events on Notifications in a particular tabContext of a browser tab. For example, show Notification, Close Notification, Listen to close notification, Listen to actions in notification.

To get the reference to NotificationContext, use the getNotificationContext function in tabContext.

Here's a Typescript example of the getnotification context from the tabContext:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext('browserTabId');
const notificationContext: INotificationContext = await tabContext.getNotificationContext('notificationId');
```

Here's a Javascript example of the getnotification context from the tabContext:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext('browserTabId');
const notificationContext = await tabContext.getNotificationContext('notificationId');
```

Supported Operations

Supported Operations

Operation Name	Description	Example
ShowNotification	Used to show a notification	Open Notification
CloseNotification	Used to close a notification.	Close Notification

Supported Events

Supported Events

Event Name	Description	Example
Listen close action event	Listens to a closer action of a notification.	Listen close action event
Listen action trigger event	Listens to the Action button click triggered in a notification.	Listen action trigger event

Supported Methods

Supported Methods

Method Name	Description	Example
publish	Publishes an operation in record context. For instance, Open Modal, Close Modal, Open Popup, Close Popup	Publish
getSupportedActions	Returns actions supported by that record context as an array.	Get supported actions
getSupportedEvents	Returns actions supported by the context of an array.	Get supported events

Note: All the notification events and operations will work only with NotificationContext.

InsightsContext

This context enables you to do actions and events insights CCA. For example, Show Insights, Dismiss Insights, Listen to dismiss Insights, Listen to actions in Insights.

You can get the reference to InsightsContext, using getInsightsContext function.

Syntax

Here's the syntax:

```
getInsightsContext(): Promise<IInsightsContext>;
```

Here's a Typescript example of get notification context from a record context:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext() as ITabContext;
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const insightContext: IInsightsContext = await recordContext.getInsightsContext();
```

Here's a Javascript example of get notification context from a record context:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
const recordContext = await tabContext.getActiveRecord();
const insightContext = await recordContext.getInsightsContext();
```

Supported Operations

Supported Operations

Operation Name	Description	Example
ShowInsights	Used to show insights.	Show Insights
DismissInsights	Used to dismiss insights.	Dismiss Insights
GetAllInsights	Used to get all insights in the select RecordContext.	Get All Insights

Supported Events

UEF supports subscribe and getSupportedEvents

Supported Events

Method Name	Description	Example
Listen Dismiss event	Listens to dismiss action of an Insight.	Show Insights
Listen Action Trigger event	Listens to Action button click triggered in Insights.	Listen Action Trigger event with details

Supported Methods

UEF supports publish and getSupportedActions

Supported Methods

Operation Name	Description	Example
publish	Publish an operation insights context.	Publish
getSupportedActions	Returns actions supported by the context as an array.	Get supported actions
getSupportedEvents	Returns actions supported by the context of an array.	Get supported events

ToolbarContext

Enables you to access the MCA toolbar for its configuration.

Syntax

Here's the syntax:

```
getToolbarContext(): Promise<IToolbarContext>;
```

Here's a Typescript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as
 IPhoneContext;
const toolBarContext: IToolbarContext = phoneContext.getToolbarContext();
```

Here's a Javascript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
```

```
const mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const toolBarContext = phoneContext.getToolbarContext();
```

Supported Operations

Supported Operations

Operation Name	Description	Example
SetToolbarProperties	Set the dimensions of the toolbar	*SetToolbarProperties*

Supported Methods

Supported Methods

Operation Name	Description
publish	Publish an operation in the toolbar context.

Generate request object using requestHelper

To subscribe to app-level events or to perform app-level operations, the external application must invoke the Subscribe or Publish APIs provided by the UI Events Framework.

These APIs will expect a request object that contains the details of the events to be listened to or operations to be performed. UEF provides a requestHelper object to generate the request object specific to each event pt operation.

Structure of the requestHelper object

```
IRequestHelper {
 createSubscriptionRequest(eventName: EventName): IEventRequest;
 createPublishRequest(operationName: OperationName): IOperationRequest;
}
IEventRequest {
 getEventName(): string;
}
IOperationRequest {
 getOperationName(): string;
}
```

Code samples

Here are examples of generating the request object for an event subscription.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(requestObject,(response: IEventResponse) => {
 // custom code
 });
```

Here's a Javascript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
globalContext.subscribe(requestObject,(response) => {
 // custom code
});
```

Here are examples of generating a request object for performing and operation.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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
 }).catch((error: IErrorData) => {
 // custom code
 });
```

Here's a Javascript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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
 });
```

Call the APIs

The following APIs are available in the context

APIs

API	Description
Subscribe API	To subscribe to an event fired from the Fusion application.
SubscribeOnce API	To subscribe to a Fusion application event just once.
Publish API	To inform the Fusion application to perform certain operations.
Dispose API	To dispose or unregister any subscriptions that are added for an event.

Subscribe API

The Subscribe API is used to subscribe to an event fired from the Fusion application.

Using this API, the external application can listen to the application, tab, or object level events from the Fusion application. Once an event is subscribed using this API, the external application will be notified until the subscription is disposed.

Syntax

```
subscribe: (payload: IEventSubscriptionPayload, callbackFunction: (response:IEventResponsePayload) => void)
 => ISubscriptionContext;
```

Parameters

Parameter Name	Required?	Description
payload	Yes	Request details for the subscription with corresponding event name.
callbackFunction	Yes	A callback function, that will be called whenever the event is triggered inside the Fusion application.

SubscribeOnce API

The SubscribeOnce API can be used to subscribe to a Fusion application event just once.

Using this API, external applications can listen to the application, tab, or object level events from the Fusion application. But this subscription will be disposed of automatically after the first notification.

Syntax

```
subscribeOnce: (payload: IEventSubscriptionPayload, callbackFunction: (response: IEventResponsePayload) =>
 void) => ISubscriptionContext;
```

Parameters

Parameters

Parameter Name	Required?	Description
payload	Yes	Request details for the subscription with corresponding event name.
callbackFunction	Yes	A callback function, which should be getting called whenever the event is triggered inside the Fusion application.

Publish API

The Publish API informs the Fusion application to perform certain operations.

The API will return a promise to which we can add a then-catch block. The then block would return the operation's status and any data returned after the process is completed, and the catch block will give the error details if the operation isn't executed successfully.

Syntax

```
publish: (payload: IOperationPublishPayload) => Promise<IOperationResponsePayload>;
```

Parameters

Parameters

Parameter Name	Required?	Description
payload	Yes	Request details for the subscription with corresponding event name.

Dispose API

The Dispose API can dispose or unregister any subscriptions that are added for an event.

By disposing the subscription, the API will not trigger any further notifications to the client side receiver.

Syntax

```
dispose:() => void;
```

Supported Events

The events generated by the UI Events Framework are classified into the following divisions:

- Controllable Events
- Non Controllable Events

Controllable Events

You've the option of cancelling or deferring these events for a certain period of time.

Controllable Events

Event	Context	Notes
OnBeforeSave	RecordContext	This event is fired before firing the API request to commit the data to the server. End-users can do asynchronous operations in this callback or Cancel this event. This event can wait for an asynchronous operation to be completed.
onToolbarAgentCommand	PhoneContext	During a phone call scenario, if users want to control this event by either resolving or rejecting it, they must pass a promise as the

Event	Context	Notes
		return of this event callback function. On notification of this event subscription, the user can check the command, for example, getActiveInteractionCommands, in the event response and pass outData for that command by resolving the promise with that outData. The event subscription should be added only once, during initialization of the MCA code. The user will get this event notification only once for a session.
onToolbarInteractionCommand	PhoneContext	During a phone call scenario if users want to control this event by either resolving or rejecting it, they must pass a promise as the return of this event callback function. On notification of this event subscription, the user can check the command in the event response. Commands supported are, accept, disconnect, reject, setActive. These commands corresponds to accept disconnect reject operations agent might make in the Fusion application. The user can wire up their logic for each of these commands, on the callback of this event response.
OnBeforeMessageSend	RecordContext	This event will be fired before firing the REST API request to send message to recipients, from the Compose Message smart action. The user should can perform async operations in this callback and allow or cancel this message send process. This means, this event can wait for an async operation to be completed.

Non Controllable Events

You can't control these events in the same way that you can control controllable events. These events are similar to notification events as that notify the client when an event occurs in the application in a fire and forget mechanism.

Events	Context	Notes
TabOpen	GlobalContext	This event will be fired when a browser tab or MSI tab gets opened with the fusion application.

Events	Context	Notes
TabClose	GlobalContext	This event will be fired when a browser tab or MSI tab gets closed with the fusion application.
TabChange	GlobalContext	This event will be fired when a browser tab or MSI tab happens with the fusion application.
CustomEvent	GlobalContext TabContext RecordContext	This event can be used to support any business specific custom events.
ContextOpen	TabContext	This event will be fired when an object is opened in UI. For example: SR open.
DataLoad	RecordContext	This event will be fired when data is loaded. For example, after fetching a SR object data.
ContextClose	TabContext	This event will be fired when an object is closed in UI. For example, SR close.
FieldValueChange	RecordContext	This event will be fired when a field value change occurs. For example, SR title field change.
OnAfterSave	RecordContext	This event will be fired after saving a open object in UI. For example, After saving an SR.
SidePaneOpen	SidePaneContext	This event will be fired when a SidePane is opened.
SidePaneClose	SidePaneContext	This event will be fired when a SidePane is closed.
onDataUpdated	PhoneContext	This event is used to listen to a DataUpdate event that happens in Fusion application during a phone call scenario.
onOutgoingEvent	PhoneContext	This event is used to listen to an outBound call event that happens in Fusion application.
OnAfterMessageSen	RecordContext	This event is a notify event which will be fired after successfully sending message to recipients from Compose Message Smart Action.
NotificationClose	NotificationContext	This event will be fired when a Notification is closed.
NotificationAction	NotificationContext	This event will be fired when a Notification's action button click.

Events	Context	Notes
WindowClose	ModalWindowContext	This event will be fired when a Modal or dialog box is closed.

Supported Operations

Operation	Context	Notes
disableFeature	PhoneContext	Informs the Fusion application that a subset of available functionality must be disabled because the toolbar hasn't implemented it.
agentStateEvent	PhoneContext	Notifies the Fusion application of a change in the user's signed in or availability status for the specified channel
readyForOperation	PhoneContext	Notifies the Fusion application that the toolbar is ready for operation.
outboundCommError	PhoneContext	The toolbar can publish this API to notify Fusion Application that an error occurred during initiation of the outbound event. The error occurs if the identifier of the event, such as phone number, or email can't be used to establish a connection.
CustomEvent	GlobalContextTab TabContext Record Context	This event can be used to support any business specific custom events.
closeCommEvent	PhoneContext	This action is required to publish in a call decline scenario. When an agent clicks Decline on a call notification, the CTI integration code base receives a notification for the onAgentInteractionCommand event with command as **`decline`**if it has a subscription added for the same.
startCommEvent	PhoneContext	This action is required to publish when there's a call scenario in the newCommEvent. For example, in an inbound call scenario, a user side listens for onToolbarInteractionCommand and when a notification is received with the command as accept the user publishes this action to establish that call connection.

Operation	Context	Notes
newCommEvent	PhoneContext	This action is required to publish when there's a ring received scenario. For example, when a user is trying to call an agent, the Fusion application should show a notification with the Accept or Decline Button. To enable this, the MCA integrated CTI should publish a newCommEvent request.
SetComposeMessageData	RecordContext	This operation is used to set the message data of the Compose Message Action panel which is opened through compose smart actions available on a record. It enables a user to set CC, BCC , To , Message Subject and Message Body of the compose message action panel. It also allows users to set custom fields, in the compose action panel, if they've been added through extensibility.
CloseModal	ModalWindowContext TabContext	To close a modal window.
Close Notification	NotificationContext	To close a Notification.
CloseTab	TabContext	To close a particular MSI tab or browser tab.
ClosePopup	ModalWindowContext TabContext	To close a dialog box.
ExpandSidePane	SidePaneContext	To expand a SidePane.
GetFieldValue	RecordContext	To fetch the current value of a field. For example, to get the current value of the Title field of an SR.
GetAgentInfo	GlobalContext TabContext	To fetch the information about current logged in user.
FocusTab	TabContext	To focus a particular MSI tab or browser tab.
getConfiguration	PhoneContext	To get configuration information that enables the toolbar to evaluate the features supported by the Fusion application.
InvokeServiceConnection	GlobalContext	To call a service defined in application and fetch its response.
OpenModal	ModalWindowContext TabContext	To open a modal window.

Operation	Context	Notes
PopOperation	GlobalContext TabContext	To open a new browser tab or MSI tab or any custom page in any desired tab.
OpenPopup	ModalWindowContext TabContext	To open a dialog box.
Save Record	RecordContext	To save an open record. For example, save an SR form from edit page or save an SR from the create form.
Show Notification	NotificationContext	To show a notification.
UpdateSidePane	SidePaneContext	To update an existing side pane (setVisibility, setIcon and setSectionId)
SetFieldValue	RecordContext	To update the value of a particular field. For example, update the Title field of an SR.
CollapseSidePane	SidePaneContext	To collapse a SidePane
SetChannelAvailability	GlobalContext	Enabled or disable the phone icon.
SetToolbarProperties	PhoneContext	To set the dimensions of toolbar.
ExecuteSmartAction	SmartActionContext	To execute a SmartAction.

Supported Operations

You can use extensibility operations to execute certain tasks in the Fusion application. Supported operations are shown in the following table:

Supported Operations

Operation	Context	Notes
PopOperation	GlobalContext Tab Context	Use to pop a new browser tab or MSI tab or any custom page in any desired tab.
FocusTab	TabContext	Use to focus a particular MSI tab or browser tab.
CloseTab	TabContext	Use to close a particular MSI tab or browser tab.

Operation	Context	Notes
InvokeServiceConnection	GlobalContext	Use to call a service defined in application and fetch its response.
SetFieldValue	RecordContext	Use to update the value of a particular field. , For example, update the Title field of an SR.
GetFieldValue	RecordContext	Use to fetch the current value of a field. For example, to get the current value of the Title field of an SR.
SaveRecord	RecordContext	Use to save an open record. For example, to save an SR form from the edit page or save an SR from the create form.
CustomEvent	GlobalContext TabContext RecordContext	Use this event to support any business-specific custom events.
GetAgentInfo	GlobalContext TabContext	Use to fetch the information about current logged in user.
OpenModal	ModalWindowContext TabContext	Use to open a modal window.
CloseModal	ModalWindowContext TabContext	Use to close a modal window.
OpenPopup	ModalWindowContext TabContext	Use to open a dialog box.
ClosePopup	ModalWindowContext TabContext	Use to close a dialog box.
ExpandSidePane	SidePaneContext	Use to expand a SidePane.
CollapseSidePane	SidePaneContext	Use to collapse a SidePane.
UpdateSidePane	SidePaneContext	Use to update an existing side pane (setVisibility, setIcon and setSectionId).

3 Example Use Cases

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 Fusion 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');

 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', 'v1');
 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 Fusion 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 Fusion 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();
 });
```

ContextOpen Event

This event is started 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', 'V1');
 const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
 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', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
tabContext.subscribe(requestObject, (message) => {
 const response = message;
 const recordContext = response.getResponseData();
});
```

Context Close Event

This event is started 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', 'V1');
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());
});
```

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', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
tabContext.subscribe(requestObject, (message) => {
 const response = message;
 const context = response.getResponseData();
 console.log(context.getObjectType());
 console.log(context.getObjectId());
});
```

DataLoad Event

This event is started 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', 'v1');
 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', 'v1');
```

```
 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 are examples of subscribing to FieldValueChange event where field names are passed.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
 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 started 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
 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 started before 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
 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 call 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
```

```
 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
 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:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest opening an Edit Service Request page with the detail view:

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest used to open and Create a Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest opening an Edit Case page in a new browser tab:

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest used to open a Create Contact page.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest used to open an Edit Contact page.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest used to open a Create Account page.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of IPopFlowInAppRequest used to open an Edit Account page.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of the IPopFlowAppUIRequest used to open 360 pages.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples of the IPopFlowGenericRequest to open the Article page.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 are examples for IPopFlowGenericRequest to open the Article page (full URL including https):

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 shows examples of Pop with a Set operation.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 shows examples of Pop with a Get operation.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
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', 'v1');
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 Required Field and Save Operation

The following shows examples of Pop with the Set operation and Save operation.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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();
    //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: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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
});
```

Pop native svc-contact page in the Fusion application

In the Fusion application, the svc-contact page is the native page for opening the Contact detail view or the create a new contact page.

A page parameter named selectedView is used to render the Create Contact or Contact Detail views in the svc-contact page. If the value of the selectedView page parameter is createContact then it will render the create contact view. To open the detail view of an existing contact, the value of selectedView page parameter should be set as defaultView and the value of SVCMCA_CONTACT_NUMBER page parameter which must be set as the ID of the contact which is to be opened.

You access the pop native svc-contact page in the Fusion application, by opening the Create Contact view.

Here's the Javascript code for Pop svc-contact - create contact view:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
payload.setFlow('sr');
payload.setPage('svc-contact');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "createContact"});
const popResponse = await globalContext.publish(payload);
const tabContext = popResponse.getResponseData();
const contactRecord = await tabContext.getActiveRecord();
var sfvPayload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
sfvPayload.field().setValue("Contact.FirstName", "User123");
await contactRecord.publish(sfvPayload);
```

Here's the Typescript code for Pop svc-contact - create contact view:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const payload: IPopFlowAppUIRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowAppUIRequest;
payload.setFlow('sr');
payload.setPage('svc-contact');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "createContact"});
const popResponse: IPopFlowResponse = await globalContext.publish(payload) as IPopFlowResponse;
const tabContext: ITabContext = popResponse.getResponseData() as ITabContext;
const contactRecord: IRecordContext = await tabContext.getActiveRecord() as IRecordContext;
```

```
var sfvPayload: ISetFieldValueOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest;
sfvPayload.field().setValue("Contact.FirstName", "User123");
await contactRecord.publish(sfvPayload);
```

Pop native svc-contact page in the Fusion Application Open Contact detail view

Here's the Javascript code for POPing the svc-contact page (for an existing contact) in an MSI tab:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
payload.setFlow('sr');
payload.setPage('svc-contact');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "defaultView", "SVCMCA_CONTACT_NUMBER": "CDRM_948047"});
const popResponse = await globalContext.publish(payload);
const tabContext = popResponse.getResponseData();
const contactRecord = await tabContext.getActiveRecord();
```

Here's the Typescript code for POPing the svc-contact page (for an existing contact) in an MSI tab:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const payload: IPopFlowAppUIRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowAppUIRequest;
payload.setFlow('sr');
payload.setPage('svc-contact');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "defaultView", "SVCMCA_CONTACT_NUMBER": "CDRM_948047"});
const popResponse: IPopFlowResponse = await globalContext.publish(payload) as IPopFlowResponse;
const tabContext: ITabContext = popResponse.getResponseData() as ITabContext;
const contactRecord: IRecordContext = await tabContext.getActiveRecord() as IRecordContext;
```

Pop svc-interaction page in the Fusion application

In the Fusion application, the svc-interaction page can also be used for opening the Contact detail view or the **Create New Contact** form.

A page parameter named selectedView is used to render the Create Contact or Contact detail views in svc-interaction page. If the value of the selectedView page parameter is createContact then it will render the create contact view. To open the detail view of an existing contact, the value of selectedView page parameter must be set as defaultView and the value of SVCMCA_CONTACT_NUMBER page parameter must be set as the ID of the contact to be opened.

You access the Pop svc-interaction page in the Fusion application through the Create contact view.

Here's the Javascript code for POPing svc-interaction page (for create contact) in MSI tab:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
payload.setFlow('sr');
payload.setPage('svc-interaction');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "createContact"});
const popResponse = await globalContext.publish(payload);
const tabContext = popResponse.getResponseData();
const contactRecord = await tabContext.getActiveRecord();
var sfvPayload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
sfvPayload.field().setValue("Contact.FirstName", "User123");
await contactRecord.publish(sfvPayload);
```

Here's the Typescript code for POPing svc-interaction page (for create contact) in MSI tab:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const payload: IPopFlowAppUIRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowAppUIRequest;
payload.setFlow('sr');
payload.setPage('svc-interaction');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "createContact"});
const popResponse: IPopFlowResponse = await globalContext.publish(payload) as IPopFlowResponse;
const tabContext: ITabContext = popResponse.getResponseData() as ITabContext;
const contactRecord: IRecordContext = await tabContext.getActiveRecord() as IRecordContext;
var sfvPayload: ISetFieldValueOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest;
sfvPayload.field().setValue("Contact.FirstName", "User123");
await contactRecord.publish(sfvPayload);
```

Pop svc-interaction page in Service Center or the Fusion Application - Open contact detail view

Here's the Javascript code for svc-interaction page (for existing contact) in MSI tab:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
payload.setFlow('sr');
payload.setPage('svc-interaction');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "defaultView", "SVCMCA_CONTACT_NUMBER": "CDRM_948047"});
const popResponse = await globalContext.publish(payload);
const tabContext = popResponse.getResponseData();
const contactRecord = await tabContext.getActiveRecord();
```

Here's the Typescript code for POPing svc-interaction page (for existing contact) in MSI tab:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const payload: IPopFlowAppUIRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowAppUIRequest;
payload.setFlow('sr');
payload.setPage('svc-interaction');
payload.setApplicationUIName('service');
payload.setInputParameters({"selectedView": "defaultView", "SVCMCA_CONTACT_NUMBER": "CDRM_948047"});
```

```
const popResponse: IPopFlowResponse = await globalContext.publish(payload) as IPopFlowResponse;
const tabContext: ITabContext = popResponse.getResponseData() as ITabContext;
const contactRecord: IRecordContext = await tabContext.getActiveRecord() as IRecordContext;
```

Pop svc-contact page in MSI tab (MSI mode), by passing entity type

Here's the Typescript code for opening svc-contact page in MSI tab (MSI mode), by passing entity type:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const payload: IPopFlowInAppRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowInAppRequest;
payload.setRecordType('Contact');
payload.setRecordId('10026016');
const popResponse: IPopFlowResponse = await globalContext.publish(payload) as IPopFlowResponse;
const tabContext: ITabContext = popResponse.getResponseData() as ITabContext;
const contactRecord: IRecordContext = await tabContext.getActiveRecord() as IRecordContext;
```

Here's the Javascript code for opening svc-contact page in MSI tab (MSI mode), by passing entity type:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('PopOperation');
payload.setRecordType('Contact');
payload.setRecordId('10026016');
const popResponse = await globalContext.publish(payload);
const tabContext = popResponse.getResponseData();
const contactRecord = await tabContext.getActiveRecord();
```

Note: Remove the entityId setter in request payload, to pop Contact Create tab in MSI mode.

Pop knowledge article in Service Center or a Fusion application in MSI tab

Here's the Typescript code for opening article page in MSI tab, by passing page parameters from application console:

```
 const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
const frameworkProvider: IUiEventsFrameworkProvider = await uefProvider.uiEventsFramework.initialize('FVC');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
const payload: IPopFlowAppUIRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
 as IPopFlowAppUIRequest;
payload.setFlow('sr');
payload.setPage('view-article');

payload.setInputParameters({
stripeCode: 'ORA_SVC_CRM',
answerId: '10026016',
});
const response: IPopFlowResponse = await tabContext.publish(payload) as IPopFlowResponse;
```

Here's the Javascript code for opening article page in MSI tab, by passing page parameters from application console:

```
const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
 const frameworkProvider = await uefProvider.uiEventsFramework.initialize('FVC');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 payload.setFlow('sr');
 payload.setPage('view-article');

 payload.setInputParameters({
 stripeCode: 'ORA_SVC_CRM',
```

```
answerId: '10026016',
});
const response = await tabContext.publish(payload);
```

Pop article page in MSI tab by passing an entity

Here's the Typescript code for opening article page in MSI tab, by passing entity:

```
const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
const frameworkProvider: IUiEventsFrameworkProvider = await
uefProvider.uiEventsFramework.initialize('FVC');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
const payload: IPopFlowInAppRequest = frameworkProvider.requestHelper.createPublishRequest('PopOperation')
as IPopFlowInAppRequest;
payload.setRecordType('Article')
payload.setRecordId('10026016')
const response: IPopFlowResponse = await tabContext.publish(payload) as IPopFlowResponse;
```

Here's the Javascript code for opening article page in MSI tab, by passing entity:

```
const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
const frameworkProvider = await uefProvider.uiEventsFramework.initialize('FVC');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
payload.setRecordType('Article')
payload.setRecordId('10026016')
const response = await tabContext.publish(payload);
```

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','V1');
 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', 'v1');
 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', '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');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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
 });
```

Publish Get Fields In UI Operation

This API is exposed on recordContext and it's used to get the list of rendered fields in UI. getFieldsinUi API returns fields available at the time of calling (because fields can be rendered dynamically).

Here's a Typescript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const requestObject: IOperationRequest =
 this.frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldsInUI') as IOperationRequest;
recordContext.publish(requestObject).then((response: IOperationResponse) => {
 const getFieldsInUIResponseData: IGetFieldsInUIResponseData = (response as
 IGetFieldsInUIResponse).getResponseData();
 console.log(getFieldsInUIResponseData.getFieldsList())
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject = this.frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldsInUI');
recordContext.publish(requestObject).then((response) => {
 const getFieldsInUIResponseData = response.getResponseData();
 console.log(getFieldsInUIResponseData.getFieldsList())
});
```

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
```

```
{
  "Resources": {
    "serviceRequests": {
      "discrColumnType": false,
      "title": "Service Request",
      "titlePlural": "Service Requests",
      "group": "OEC",
      "usage": "BusinessObject",
      "attributes": [
        {
          "name": "Title",
          "type": "string",
          "updatable": true,
          "mandatory": true,
          "inputRequired": true,
          "queryable": true,
          "allowChanges": "always",
          "precision": 400,
          "title": "Title",
          "maxLength": "400",
          "properties": {
            "DISPLAYWIDTH": "40",
            "FIELDORDER": "0.0"
          },
          "annotations": {
            "description": "The brief title of the service request."
          }
        },
        {
          "name": "SeverityCd",
          "type": "string",
          "updatable": true,
          "mandatory": false
```

Here the metadata of 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
```

```
eg: ServiceRequest.Title
```

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.

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 must call the Publish API on top of record context ito 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', 'v1');
 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', 'v1');
 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 must call the publish API on top of record context 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', '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');
 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', 'v1');
 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 are examples of subscribing to FieldValueChange event where field names are passed.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 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:

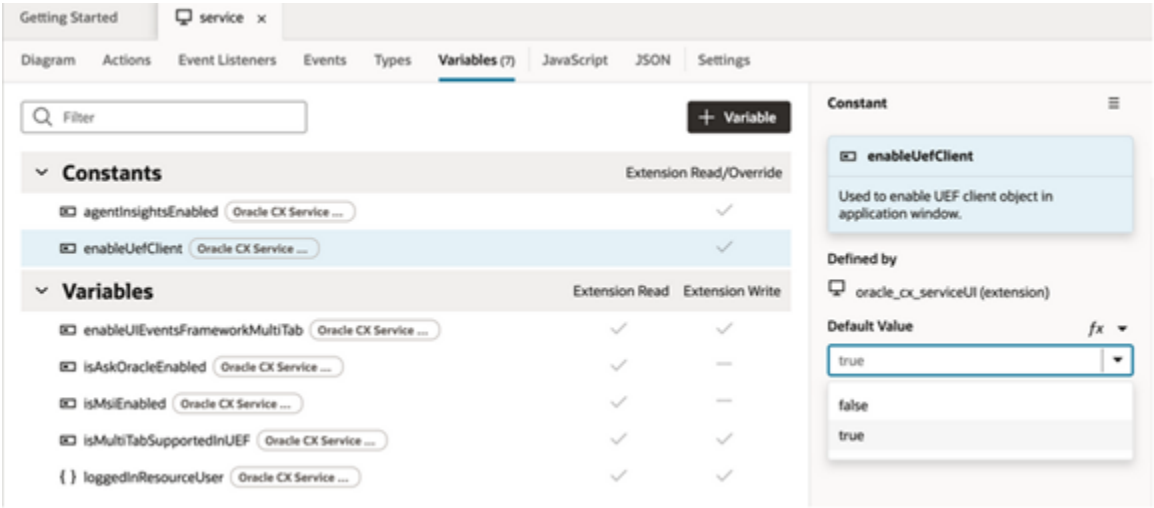
```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
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 Fusion application window

enableUefClient is an application level variable that can be exposed in the Fusion application.

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.



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 the Fusion application to access it. This feature coexists with UEF objects consumed inside external applications.

Here's a JavaScript sample added in the Fusion application 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 Fusion Application Side Pane

The Fusion application includes an extensible variable called `sidePaneButtons`. in service center, 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 newly created sidePane icons by creating sections in the extensible dynamic container available in container-page of the Fusion application.

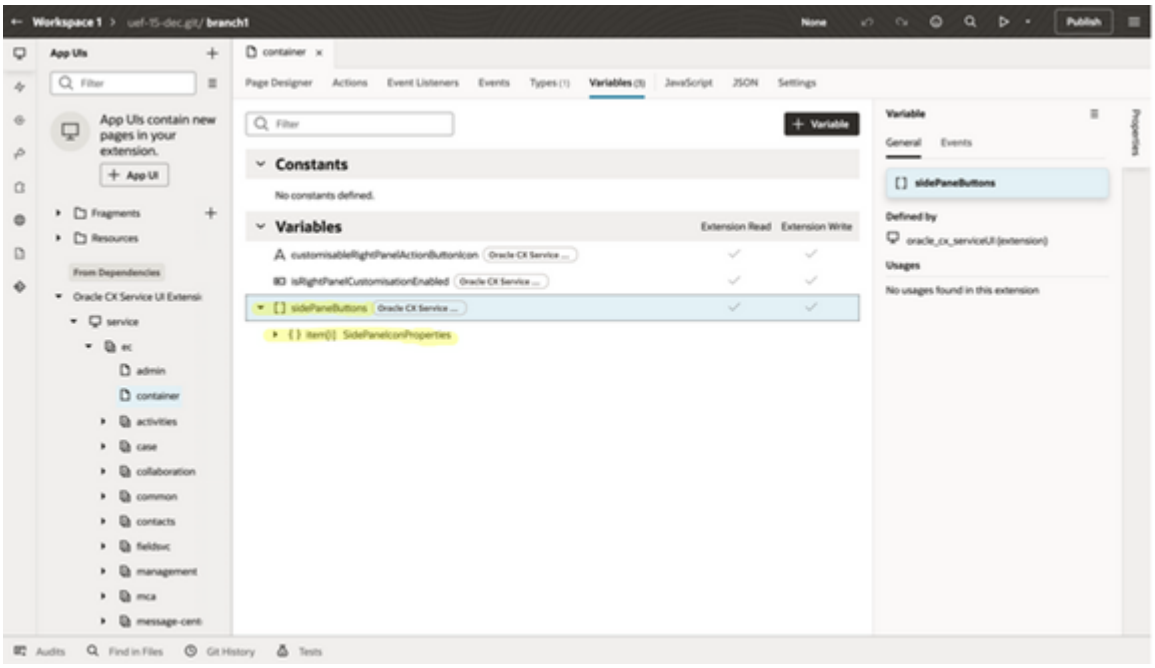
Note: The `sectionId` that you create section must match the sectionId you provide in the `sidePaneButtons` variable value.

Overview of steps to create a SidePane icon named UEF and MCA

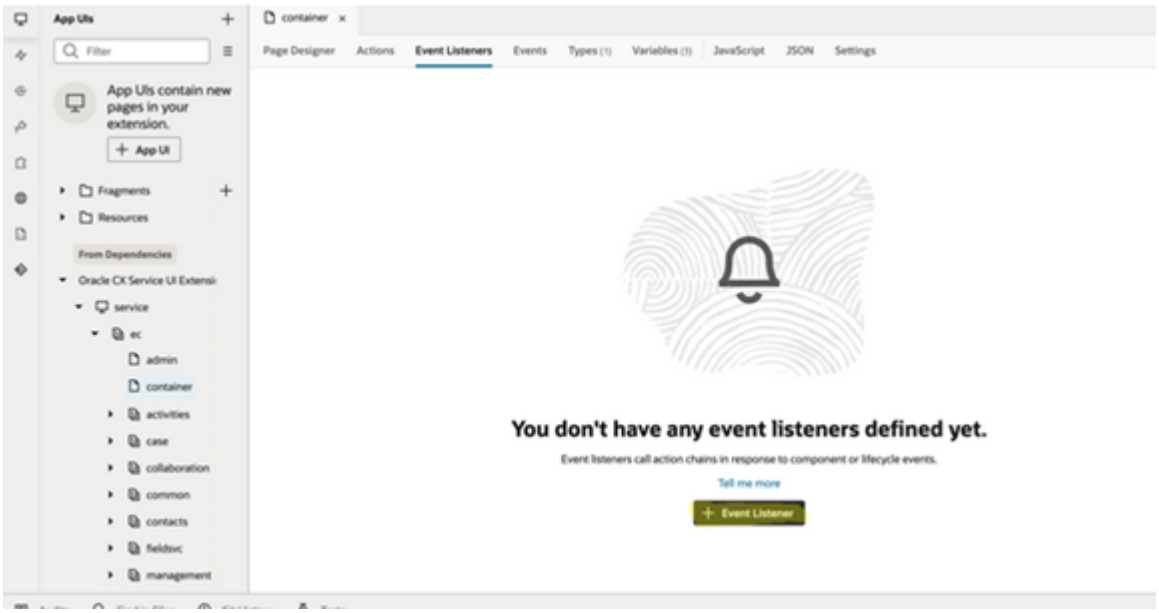
To enable the Side Pane, set the isRightPanelCustomisationEnabled variable to TRUE in the container. This enables the basic side pane.

To enable the advanced Side Pane, set the showAdvancedSidePane variable to TRUE in the container page, **along with enabling the isRightPanelCustomisationEnabled variable**.

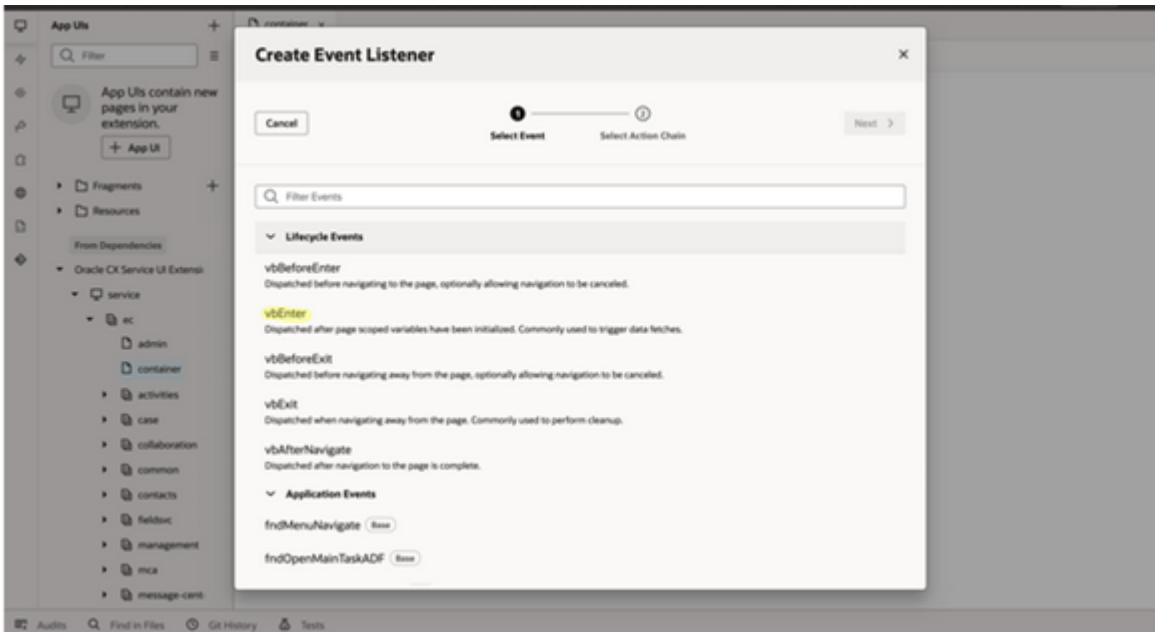
1. In to VB Studio add an array of the Sidepane icon to the SidepaneButtons variable in the container page.



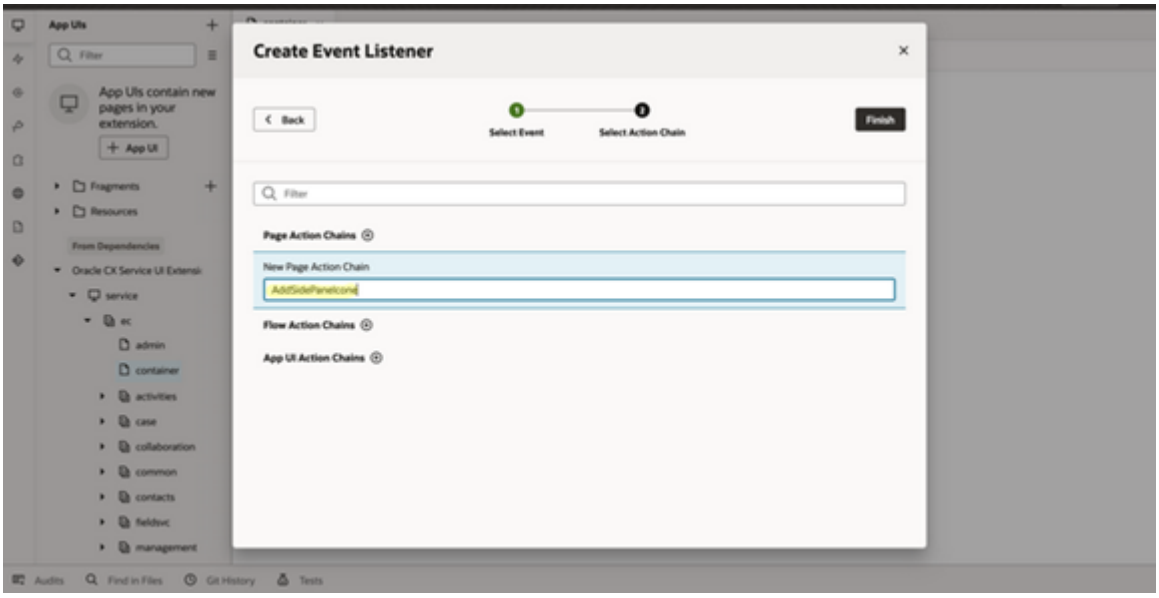
2. Create a new event listener from container page.



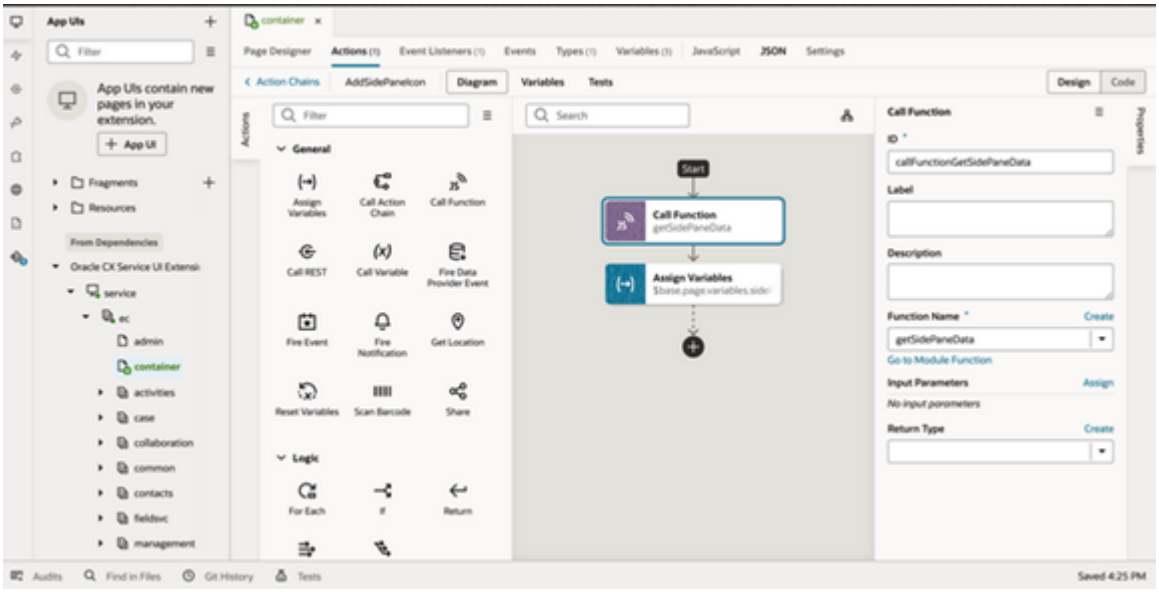
3. Create a new action chain. We're using VBEnter in this example.



4. Create a new action chain for the event listener.

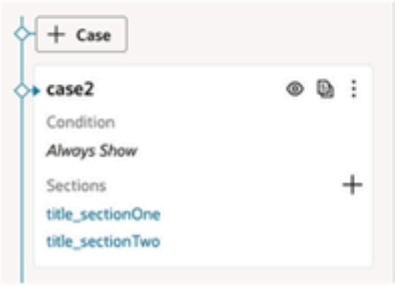


5. Add a function and assign a variable to the action chain.



JSON

```
"addSubLayouts": {
    "/extensionLayout": {
      "case2": {
        "layout": {
          "displayProperties": [
            "sectionid_sectionOne",
            "sectionid_sectionTwo"
          ],
          "sectionTemplateMap": {
            "sectionid_sectionOne": "sectionOne_template",
            "sectionid_sectionTwo": "sectionTwo_template"
          }
        }
      }
    }
  },
  "addTemplates": {
    "sectionOne_template": {
      "title": "title_sectionOne",
    },
    "sectionTwo_template": {
      "title": "title_sectionTwo",
    }
  }
```

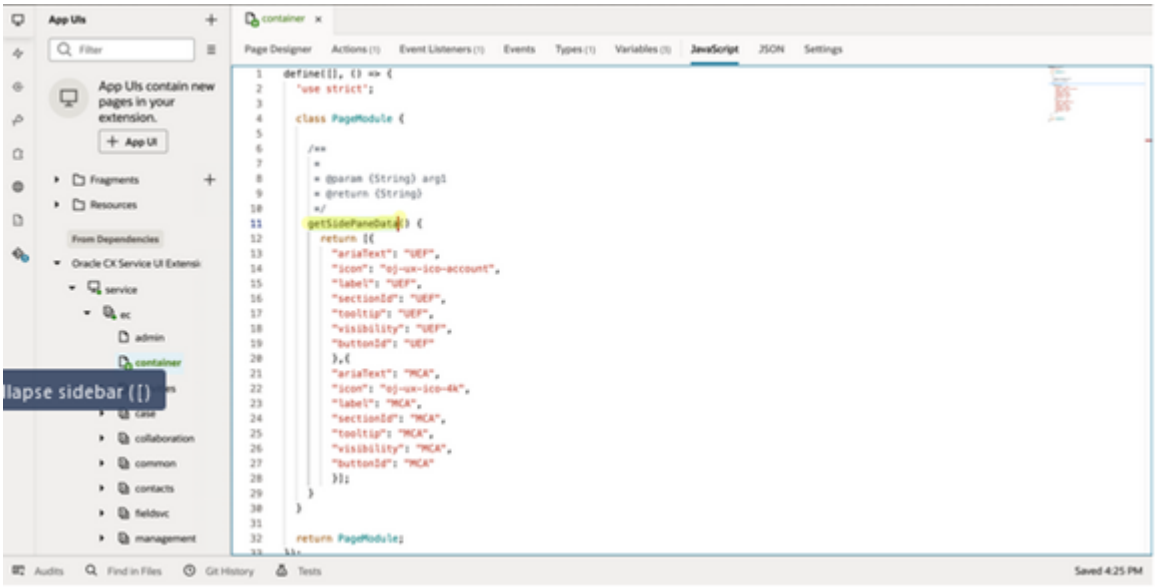


JavaScript

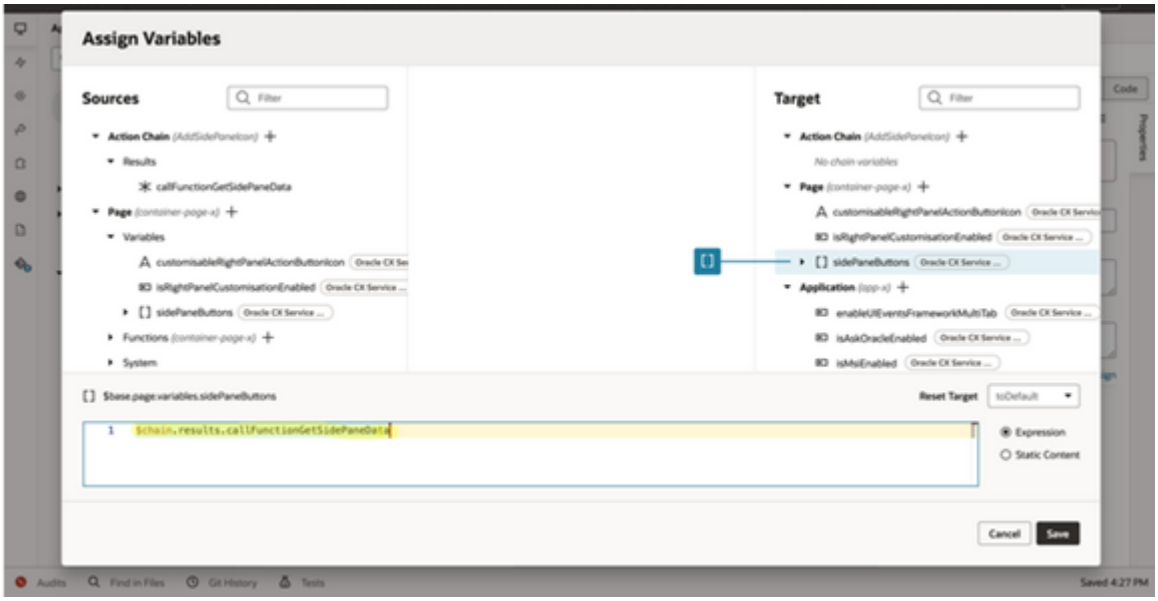
```
getSidePaneList(arg1) {
      return [
        {
          "label": "label One",
          "tooltip": "tooltip One",
          "ariaText": "ariaText One",
          "icon": "oj-ux-ico-alarm-clock",
          "sectionId": "sectionid_sectionOne",
          "visibility": true,
          "buttonId": "buttonOne"
        },{
          "label": "label Two",
          "tooltip": "tooltip Two",
          "ariaText": "ariaText Two",
          "icon": "oj-ux-ico-alarm-clock",
          "sectionId": "sectionid_sectionTwo",
          "visibility": true,
          "buttonId": "buttonTwo"
        }
      ];
}
```

```
<template id="sectionOne_template">
  <h22>section one</h22>
</template>

<template id="sectionTwo_template">
  <h22>section two</h22>
</template>
```

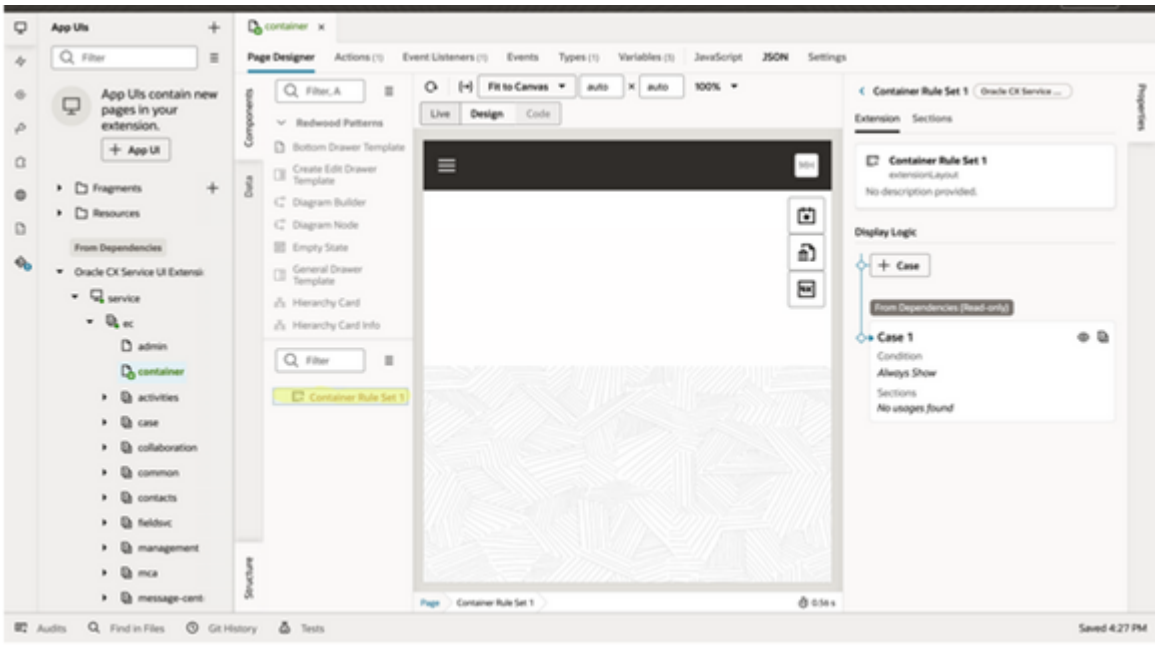
6. Note how the getSidepaneData function returns the sidepane details list.



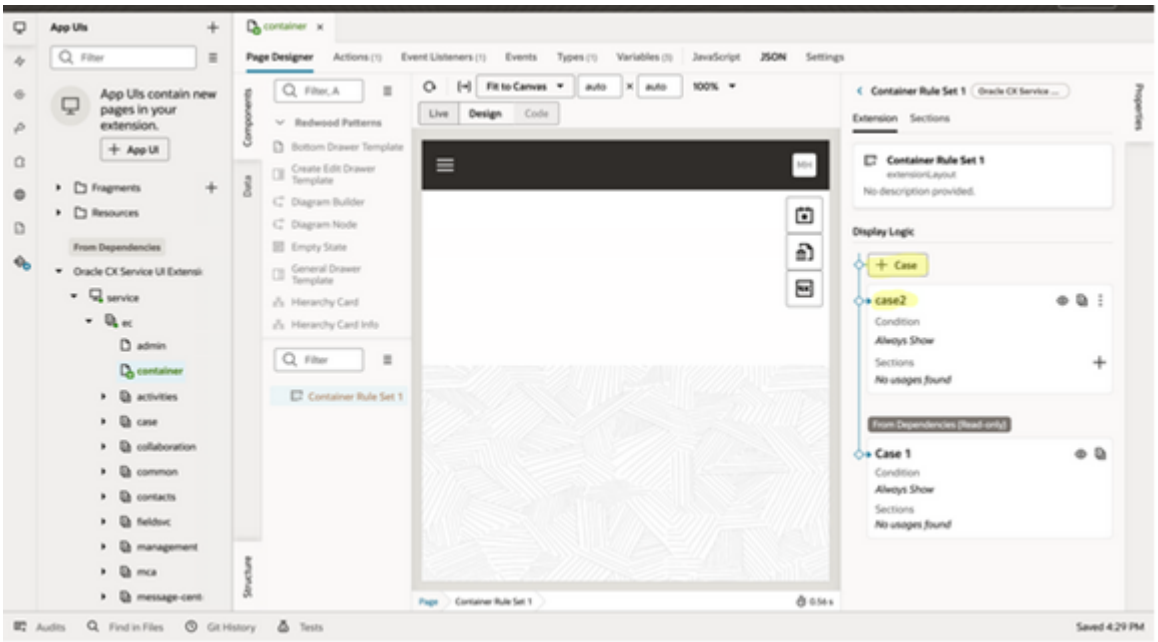
7. Use the Assign Variables screen to assign result of getSidepaneData function to the SidepaneButtons variable.



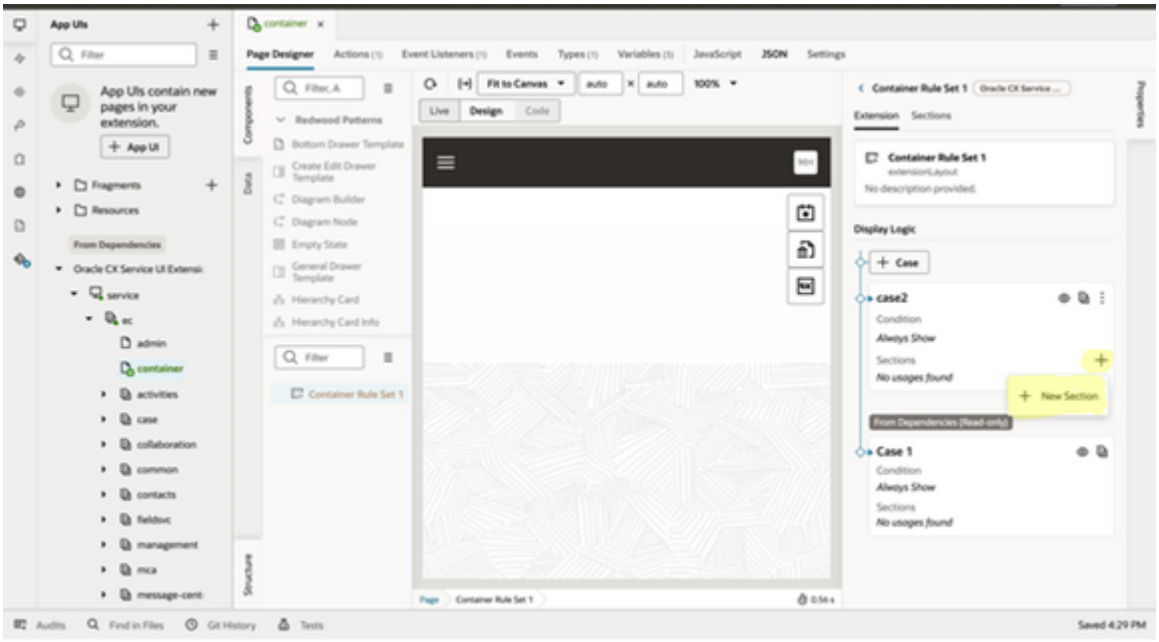
8. To add content to each sidepane item, click the Container rule set 1 from the Page Designer of the container page.



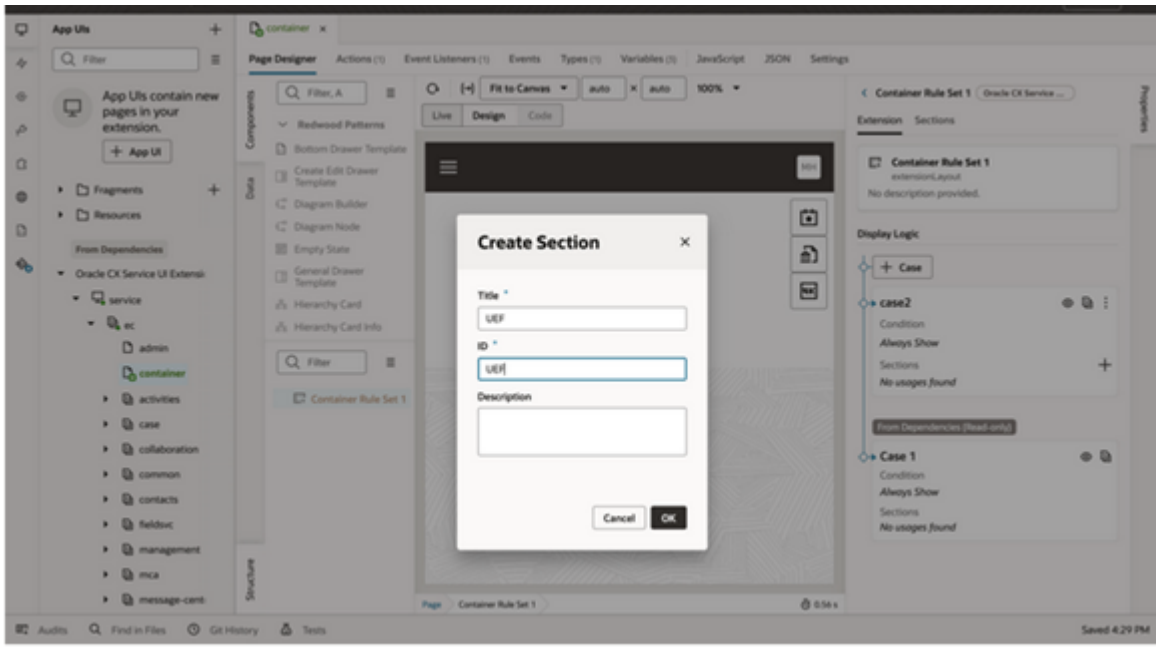
9. Now create a new case.



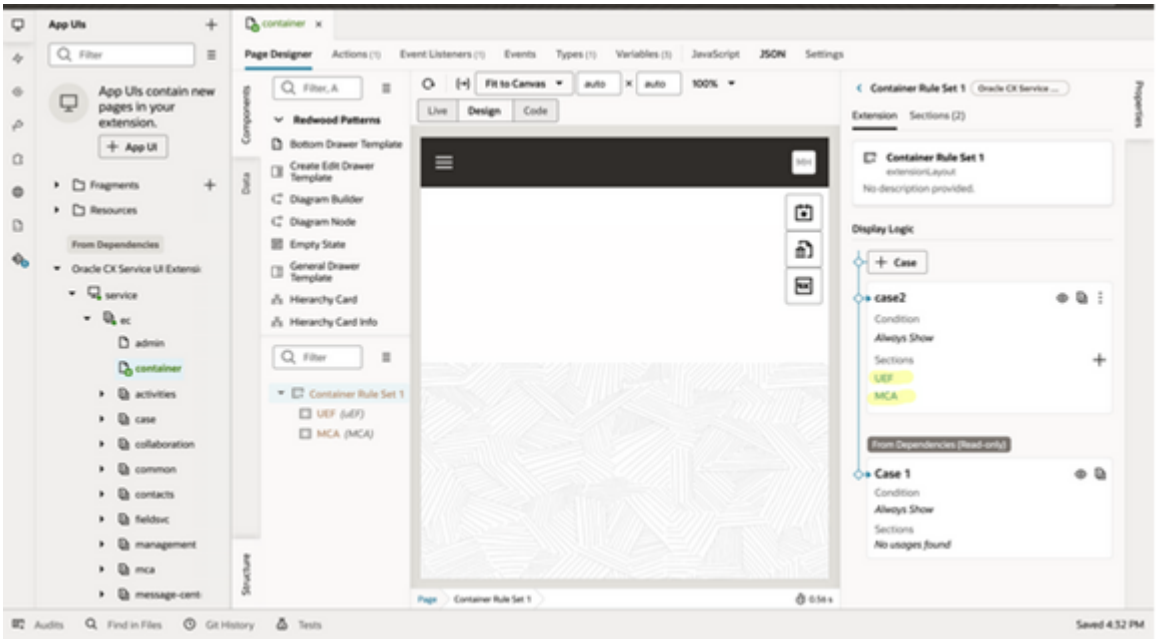
10. Create a new section from the newly created case.



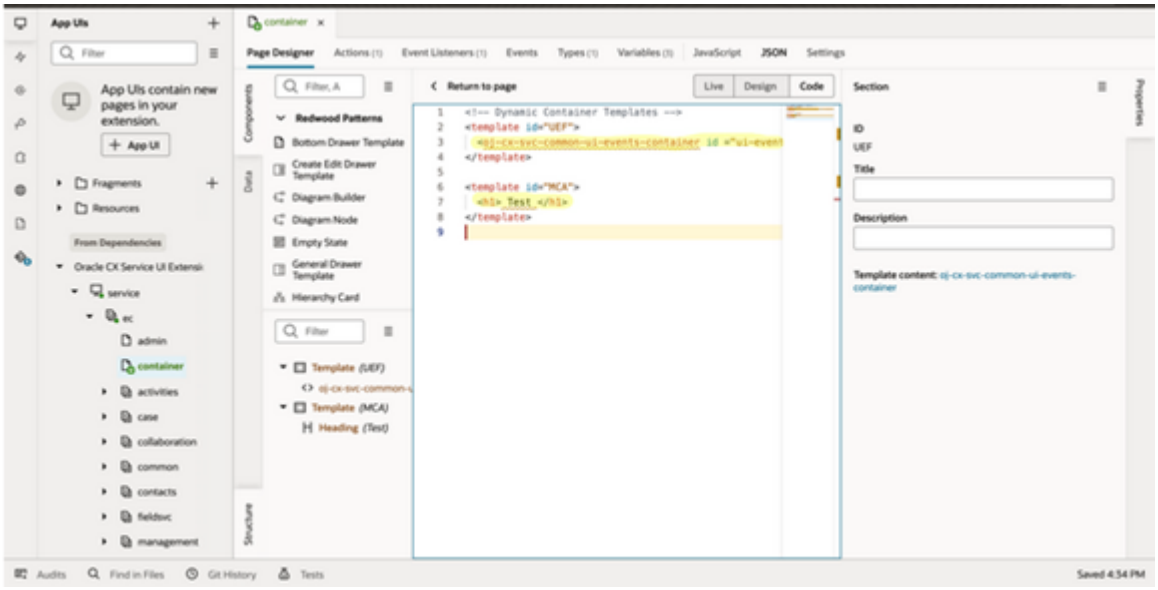
11. Create a new section called UEF for the UI Events Container.



12. Create two new sections for UEF and MCA.



13. Click the newly created section and add content from each of the sections.



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 the Fusion application. 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 = 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 the Fusion application. 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 the Fusion application. 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 the Fusion application. 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 the Fusion application. 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 the Fusion application. 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 and Events

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 action in Modal Window Context*
- *CloseModal action in Modal Window Context*
- *OpenPopup action in Modal Window Context*
- *ClosePopup action Modal Window Context*
-

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

Model Window Events

Listen to close event of with Reason for Modal Window

We can listen close event from GlobaContext and tabContext with TabId only.

1. PROGRAMMATIC if the notification closed by close notification action from UEF.
2. MANUAL if the notification closed by clicking the close icon in the notification .

Here's a Typescript example:

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

    const requestObject: IWindowSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('OnModalCloseAction') as
 IWindowSubscriptionRequest;
    requestObject.setId('modal1');
    modalWindowContext.subscribe(requestObject, (response: IEventResponse) => {
      let responseData = response as IWindowCloseActionEventResponse;
      console.log((responseData.getResponseData() as IWindowCloseActionData).getWindowId());
      console.log((responseData.getResponseData() as IWindowCloseActionData).getReason());
```

```
        });
```

Here's a JavaScript example:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
     const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
     const requestObject =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('OnModalCloseAction');
     requestObject.setId('modal1');
     modalWindowContext.subscribe(requestObject, (response) => {
         console.log(response.getResponseData().getWindowId());
         console.log(response.getResponseData().getReason());
      });
```

Listen to close event of the notification with Reason for popup window

We can listen close event from GlobaContext and tabContext with TabId and MSITabContext

1. PROGRAMMATIC if the notification closed by close notification action from UEF.
2. MANUAL if the notification closed by clicking the close icon in the notification.
3. TABCLOSE if the popup closed by close of MSITab (only for Popup in MSITab).

Here's a Typescript example:

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

  const requestObject: IWindowSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('OnPopupCloseAction') as
 IWindowSubscriptionRequest;
  requestObject.setId('popup1');
  modalWindowContext.subscribe(requestObject, (response: IEventResponse) => {
    let responseData = response as IWindowCloseActionEventResponse;
      console.log((responseData.getResponseData() as IWindowCloseActionData).getWindowId());
      console.log((responseData.getResponseData() as IWindowCloseActionData).getReason());
   });
```

Here's a JavaScript example:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
    const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
    const requestObject =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('OnPopupCloseAction');
    requestObject.setId('popup1');
    modalWindowContext.subscribe(requestObject, (response) => {
        console.log(response.getResponseData().getWindowId());
        console.log(response.getResponseData().getReason());
     });
```

Modal Window Operations in TabContext

Modal Window Operations in TabContext

You can perform Modal Window operations on Global Context, and browser Tab context. 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

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

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

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 in Tab Context of given TabId

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

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

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 isn't 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();
}
```

OpenPopup action in Tab Context of given TabId and msiTabId

Here's a TypeScript example:

```
const closePopup = async () => {
    const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
    const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('tabId','msiTabId');
```

```
    const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('ClosePopup') as IOpenModalWindowRequest;
    requestObject.setId('popup1');
    const response: IModalWindowOperationResponse = await tabContext.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('app',
 'v1');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('tabId');
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenPopup');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 const response = await tabContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

ClosePopup action in Tab Context of given TabId and msiTabId

Here's a TypeScript example:

```
const closePopup = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('tabId','msiTabId');
 const requestObject: IModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('ClosePopup') as IOpenModalWindowRequest;
 requestObject.setId('popup1');
 const response: IModalWindowOperationResponse = await tabContext.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('app',
 'v1');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('tabId','msiTabId');
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('ClosePopup');
 requestObject.setId('popup1');
 const response = await tabContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

OpenModal action in Modal Window Context

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();
```

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');
    requestObject.setPropagateToTabs(true);
    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');
    requestObject.setPropagateToTabs(true);
    const response = await modalWindowContext.publish(requestObject);
    const id = response.getResponseData().getId();
}
```

CloseModal action in Modal Window Context

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();
```

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');
 requestObject.setPropagateToTabs(true);
 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');
 requestObject.setPropagateToTabs(true);
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

OpenPopup action in Modal Window Context

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();
```

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.setPropagateToTabs(true);
 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.setPropagateToTabs(true);
 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 action Modal Window Context

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();
```

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');
 requestObject.setPropagateToTabs(true);
 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');
 requestObject.setPropagateToTabs(true);
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

Agent Info Operations

Agent Info Operation in Global Context

Agent Info Operation in Global Context

You can perform these actions on the Global context and the Browser Tab context.

UEF supports these Agent Info actions:

- *Get First Name of Logged-in Agent*
- *Get Last Name of Logged-in Agent*
- *Get Email Address of Logged-in Agent*
- *Get User Name of Logged-in Agent*
- *Get Party ID of Logged-in Agent*

Get First Name of Logged-in Agent

Here's a TypeScript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
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', 'v1');
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','V1');
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', 'v1');
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','V1');
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', 'v1');
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','V1');
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', 'v1');
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','V1');
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', 'v1');
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 example shows the syntax for the `getTabContext` method:

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

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', 'v1');
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', 'v1');
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

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

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

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', 'v1');
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', 'v1');
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 before 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''v1');
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''v1');
const openerContext = await frameworkProvider.getCurrentBrowserTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 openerContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example function with Current Browser TabContext

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''v1');
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''v1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example Function with Current Browser TabContext

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

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

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', 'v1');
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', 'v1');
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

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

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

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', 'v1');
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', 'v1');
const tabContext = await frameworkProvider.getTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example function with TabContext with given Tab Id

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

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

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', 'v1');
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', 'v1');
const tabContext = await frameworkProvider.getTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example Function with TabContext

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

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

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', 'v1');
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', 'v1');
const openerContext = await frameworkProvider.getTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 openerContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

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>;
```

Pop Operation in TabContext

Pop ServiceRequest Create Page using Tab context of given tab ID

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a create service request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext('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 service request page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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 Tab context

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a create service request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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();
```

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

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const openerTabContext = await frameworkProvider.getTabContext();
 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', 'v1');
 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 to open an edit Service request page with view as details view.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

```
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 are examples of IPopFlowInAppRequest used to open and Create a Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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', 'v1');
 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 edit a Case in a new browser tab.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
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', 'v1');
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', 'v1');
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 to open an Edit Contact page.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
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', 'v1');
 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 to open a Create Account page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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 to open an Edit Account page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
 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', 'v1');
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', 'v1');
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', 'v1');
 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', 'v1');
  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();
  //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 for Pop with the set operation and save.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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 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 = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const openerTabContext = await frameworkProvider.getCurrentBrowserTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = 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', 'v1');
    const openerTabContext = await  frameworkProvider.getCurrentBrowserTabContext();
    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', 'v1');
 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', 'v1');
  const openerTabContext = await  frameworkProvider.getCurrentBrowserTabContext();
  const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
  requestObject.setRecordType('ServiceRequest');
  const response = await openerTabContext.publish(requestObject);
  const tabContext = response.getResponseData();
```

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 isn'tgiven, 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 = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const openerTabContext = await frameworkProvider.getCurrentBrowserTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = 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', 'v1');
    const openerTabContext = await  frameworkProvider.getCurrentBrowserTabContext();
    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', 'v1');
 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', 'v1');
  const openerTabContext = await  frameworkProvider.getCurrentBrowserTabContext();
  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 browser TabId isn't passed.

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

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

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
```

Here's a JavaScript example the `getCurrentBrowserTabContext`.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
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', 'v1');
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', 'v1');
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);
 });
 })
 }
 })
}
```

Subscribe Custom Events

Subscribe Custom Events

You can subscribe to any custom event by calling the UEF subscribe API on GlobalContext, RecordContext and BrowserTabContext.

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 has been 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.

These examples show adding a custom event subscription on different contexts:

- *Subscribe Custom Event on GlobalContext*
- *Subscribe Custom Event on TabContext*
- *Subscribe Custom Event on RecordContext*

Subscribe Custom Event on GlobalContext

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

```
const subscribeCustomEvent = async () => {
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload: ICustomEventSubscriptionRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
 payload.setCustomEventName('customEventName');
 const globalContext: IGlobalContext = await frameworkProvider.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 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('customEventName');
 const globalContext = await frameworkProvider.getGlobalContext();
 globalContext.subscribe(payload, (response) => {
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
```

```
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Subscribe Custom Event on TabContext

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

```
const subscribeCustomEvent = async () => {
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload: ICustomEventSubscriptionRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
 payload.setCustomEventName('customEventName');
 const tabContext:ITabContext = await frameworkProvider.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 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('customEventName');
 const tabContext = await frameworkProvider.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

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','V1');
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''V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Publish Custom Events to all Browser Tabs

Publish Custom Events

You can publish any custom event by calling the UEF publish API on GlobalContext, RecordContext and Browser-TabContext.

The action name to which publish API is to be called 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 call a notification to all receivers who have before added a subscription for this same event. Also, those receivers will get the data with which this publish request is called.

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

These examples show adding a custom event publish request in different contexts:

- *Publish Custom Events*
- *Publish Custom Event on GlobalContext*
- *Publish Custom Event on TabContext*
- *Publish Custom Event on RecordContext*

Publish Custom Event on GlobalContext

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

```
const publishCustomEvent = async () => {
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload: ICustomEventRequest = frameworkProvider.requestHelper.createPublishRequest('CustomEvent') as
 ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
```

```
 const globalContext: IGlobalContext = await frameworkProvider.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:

```
const publishCustomEvent = async () => {
 const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload = frameworkProvider.requestHelper.createPublishRequest('CustomEvent');
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext = await frameworkProvider.getGlobalContext();
 globalContext.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 TabContext

Here's a TypeScript example for adding custom event publish on TabContext:

```
const publishCustomEvent = async () => {
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload: ICustomEventRequest = frameworkProvider.requestHelper.createPublishRequest('CustomEvent') as
 ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const tabContext: ITabContext = await frameworkProvider.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:

```
const publishCustomEvent = async () => {
 const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload = frameworkProvider.requestHelper.createPublishRequest('CustomEvent');
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const tabContext = await frameworkProvider.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 custom event publish on RecordContext:

```
const publishCustomEvent = async () => {
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload: ICustomEventRequest = frameworkProvider.requestHelper.createPublishRequest('CustomEvent') as
 ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext: IGlobalContext = await frameworkProvider.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:

```
const publishCustomEvent = async () => {
 const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
 const payload = frameworkProvider.requestHelper.createPublishRequest('CustomEvent');
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext = await frameworkProvider.getGlobalContext();
 globalContext.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 added on MSI TabContext will result in an error as the action isn't supported over that context.

The event subscription and publish enable you to define any new life cycle event on a record which isn't supported by UEF. A custom event subscription on a particular record context is notified 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 will be notified only when a

custom Event publish or a custom Event trigger from VB happens on that particular tab context. No other tab context subscriptions are notified.

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

Actions performed across all browser tabs

Here's a Typescript example to publish a customEvent on globalContext which can be propagated across tabs.

```
const publishCustomEvent = async () => {
 const payload: ICustomEventRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent') as ICustomEventRequest;
 payload.setPropagateToTabs(true);
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext: IGlobalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.publish(payload).then((message: any[]) => {
 const response: ICustomEventResponse = message[0] 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 to publish a customEvent on globalContext which can be propagated across tabs.

```
const publishCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent');
 payload.setPropagateToTabs(true);
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.publish(payload).then((message) => {
 console.log(message[0].getResponseData());
 console.log(message[0].getResponseData().getData());
 console.log(message[0].getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

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) to ease 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 requester. 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 Fusion 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's 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 call 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 subscribed 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. Select 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 can 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}`);
```

```
 });
}
```

Support for Virtual Fields

Set Field Value of a virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', '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.VirtualField', 'New Value');
 recordContext.publish(requestObject).then((message) => {
 const response = 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', 'v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 requestObject.field().setValue('ServiceRequest.VirtualField', 'New Value');
 recordContext.publish(requestObject).then((message) => {
 // custom code
 }).catch((error) => {
 // custom code
 });
```

Set Field Value of a referenced virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', '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.myVirtualField.Title', 'New Value');
```

```
recordContext.publish(requestObject).then((message) => {
const response = 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', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
requestObject.field().setValue('ServiceRequest.myVirtualField.Title', 'New Value');
recordContext.publish(requestObject).then((message) => {
// custom code
}).catch((error) => {
// custom code
});
```

Get Field Value of a virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const requestObject: IGetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
IGetFieldValueOperationRequest);
requestObject.setFields(['ServiceRequest.VirtualField']);

recordContext.publish(requestObject).then((message) => {
const response = message as IGetFieldValueResponse;
const titleFieldValue =
response.getResponseData().getField('ServiceRequest.myVirtualField.Title').getValue();

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

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
requestObject.setFields(['ServiceRequest.VirtualField']);
recordContext.publish(requestObject).then((response) => {
// custom code
const titleFieldValue =
response.getResponseData().getField('ServiceRequest.myVirtualField.Title').getValue();

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

```
});
```

Get Field Value of a referenced virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IGetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
 IGetFieldValueOperationRequest);
 requestObject.setFields(['ServiceRequest.myVirtualField.Title']);
 recordContext.publish(requestObject).then((message) => {
 const response = message as IGetFieldValueResponse;
 const titleFieldValue =
 response.getResponseData().getField('ServiceRequest.myVirtualField.Title').getValue();
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
 requestObject.setFields(['ServiceRequest.myVirtualField.Title']);
 recordContext.publish(requestObject).then((response) => {
 // custom code
 const titleFieldValue =
 response.getResponseData().getField('ServiceRequest.myVirtualField.Title').getValue();

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

Field value change of a virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

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

```
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();
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
requestObject.setFields(['ServiceRequest.VirtualField']);

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

Field Value change of a referenced virtual field

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const recordContext: IRecordContext = await tabContext.getActiveRecord();

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

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();
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
requestObject.setFields(['ServiceRequest.myVirtualField.Title']);

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

Notifications

Show notifications inside browser tabs and desktop notifications

Here's an example of simple toast message with title and summary.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
requestObject.setTitle('title');
requestObject.setSummary('Error message summary');
notificationContext.publish(requestObject).then((message) => {
 const response: INotificationOperationResponse = message as INotificationOperationResponse;
 //custom code
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
  let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
  const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
  requestObject.setTitle('title');
  requestObject.setSummary('error message summary ');
  notificationContext.publish(requestObject).then((response) => {
  // custom code
  }).catch((err) => {
  console.log(err);
  });
```

Show notifications in TabContext inside browser tabs and desktop notifications

Here's an example of simple toast message with title and summary.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
  const tabContext: ITabContext = await frameworkProvider.getTabContext('browserTabId');
  let notificationContext: INotificationContext = await
 tabContext.getNotificationContext('notificationId007');
  const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
  requestObject.setTitle('title');
  requestObject.setSummary('Error message summary');
  notificationContext.publish(requestObject).then((message) => {
    const response: INotificationOperationResponse = message as INotificationOperationResponse;
```

```
    //custom code
  });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle('Title');
requestObject.setClosable(false);
notificationContext.publish(requestObject).then((response) => {
// custom code
}).catch((err) => {
console.log(err);
});
```

Show notification with summary

Here's an example of showing a notification with summary. Summary will display in desktop notification also.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject: IShowNotificationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
 requestObject.setTitle('Title');
 requestObject.setSummary('Summary of the notification - displays in desktop notification too');
 notificationContext.publish(requestObject).then((message) => {
 const resp = message as INotificationOperationResponse;
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 requestObject.setTitle('Title');
 requestObject.setSummary('Summary of the notification - displays in desktop notification too');
 notificationContext.publish(requestObject).then((resp) => {
 //custom Code
 }).catch((err) => {
 console.log(err);
 });
```

Show notification without notification close icon

Here's an example of showing a notification without close button in the notification.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

```
 let notificationContext: INotificationContext = await
frameworkProvider.getNotificationContext('notificationId007');
 const requestObject: IShowNotificationRequest =
frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as     IShowNotificationRequest;

requestObject.setTitle('Title');
requestObject.setClosable(true);
notificationContext.publish(requestObject).then((message) => {
      const resp = message as INotificationOperationResponse;
  });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle(fieldName);
requestObject.setClosable(false);
notificationContext.publish(requestObject).then((response) => {
// custom code
}).catch((err) => {
console.log(err);
});
```

Show notification with notification close icon

Here's an example of showing a notification with close button in the notification.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject: IShowNotificationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
 requestObject.setTitle('Title');
 requestObject.setClosable(true);
 notificationContext.publish(requestObject).then((message) => {
 const resp = message as INotificationOperationResponse;
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 requestObject.setTitle('Title');
 requestObject.setClosable(true);
 notificationContext.publish(requestObject).then((resp) => {
 //custom Code
 }).catch((err) => {
 console.log(err);
 });
```

Show notification with AutoTimeout

Here's an example of showing a notification with autoTimeout. Notification automatically closes after given seconds.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
    let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
    const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;

requestObject.setTitle('Title');
requestObject.setAutoTimeout(10);
    notificationContext.publish(requestObject).then((message) => {
        const resp = message as INotificationOperationResponse;
    });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 requestObject.setTitle('Title');
 requestObject.setAutoTimeout(10);
 notificationContext.publish(requestObject).then((resp) => {
 //custom Code
 }).catch((err) => {
 console.log(err);
 });
```

Show notification with multiple actions

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
    let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
    const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;

    requestObject.setTitle('Title');
    requestObject.setActions([{id: 'join', name: 'Join Call'},{id: 'reject', name: 'Reject Call'}]);

    notificationContext.publish(requestObject).then((message) => {
        const resp = message as INotificationOperationResponse;
    });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
```

```
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle('Title');
requestObject.setActions([{id: 'join', name: 'Join Call'},{id: 'reject', name: 'Reject Call'}]);
notificationContext.publish(requestObject).then((resp) => {
//custom Code
}).catch((err) => {
console.log(err);
});
```

Show notification with custom icon

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
  let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
  const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as     IShowNotificationRequest;

requestObject.setTitle('Email Notification');
  requestObject.setIcon('https://www.oracle.com/webfolder/technetwork/jet-1000/content/images/email.png');

notificationContext.publish(requestObject).then((message) => {
      const resp = message as INotificationOperationResponse;
  });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 requestObject.setTitle('Email Notification');
 requestObject.setIcon('https://www.oracle.com/webfolder/technetwork/jet-1000/content/images/email.png');
 notificationContext.publish(requestObject).then((resp) => {
 //custom Code
 }).catch((err) => {
 console.log(err);
 });
```

Show notification with all details

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
    let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
    const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;

requestObject.setType('error');
requestObject.setTitle('Title');
requestObject.setAutoTimeout(8);
```

```
requestObject.setClosable(true);
requestObject.setSummary('error message summary ');
requestObject.setActions([{id: 'join', name: 'Join Call'}, {id: 'reject', name: 'Reject Call'}]);
    notificationContext.publish(requestObject).then((message) => {
        const resp = message as INotificationOperationResponse;
        if (Array.isArray(resp)) {
            resp.forEach((response, i) => {
                console.log(response.getContext().getNotificationId());
            });
        } else {
            console.log(resp.getResponseData().getNotificationId())
        }
    });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 requestObject.setType('error');
 requestObject.setTitle('Title');
 requestObject.setAutoTimeout(8);
 requestObject.setClosable(true);
requestObject.setSummary('error message summary ');
requestObject.setActions([{id: 'join', name: 'Join Call'}, {id: 'reject', name: 'Reject Call'}]);
 notificationContext.publish(requestObject).then((resp) => {
 if (Array.isArray(resp)) {
 let result;
 resp.forEach((response, i) => {
 console.log(response.getContext().getNotificationId());
 });
 } else {
 console.log(resp.getResponseData().getNotificationId())
 }
 }).catch((err) => {
 console.log(err);
 });
```

Close notification

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('CloseNotification') as IOperationRequest;
 notificationContext.publish(requestObject).then((message) => {
 const response = message as INotificationOperationResponse;
 //custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('CloseNotification');
 notificationContext.publish(requestObject).then((resp) => {
```

```
//custom code
}).catch((err) => {
//custom code
});
```

Close notification in TabContext

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext('browserTabId');
 let notificationContext: INotificationContext = await
 tabContext.getNotificationContext('notificationId007');
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('CloseNotification') as IOperationRequest;
 notificationContext.publish(requestObject).then((message) => {
 const response = message as INotificationOperationResponse;
 //custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext('browserTabId');
 let notificationContext = await tabContext.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('CloseNotification');
 notificationContext.publish(requestObject).then((resp) => {
 //custom code
 }).catch((err) => {
 //custom code
 });
```

Open notification in TabContext with given tabId and MSITabId

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext('browserTabId','msiTabId');
    let notificationContext: INotificationContext = await
 tabContext.getNotificationContext('notificationId007');
    const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
    requestObject.setTitle('Title');
    notificationContext.publish(requestObject).then((message) => {
        const response = message as INotificationOperationResponse;
        //custom code
    });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext('browserTabId','msiTabId');
```

```
let notificationContext = await tabContext.getNotificationContext('notificationId007');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle('Title');
notificationContext.publish(requestObject).then((resp) => {
//custom code
}).catch((err) => {
//custom code
});
```

Close notification in TabContext with given tabId and MSITabId

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext('browserTabId','msiTabId');
 let notificationContext: INotificationContext = await
 tabContext.getNotificationContext('notificationId007');
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('CloseNotification') as IOperationRequest;
 notificationContext.publish(requestObject).then((message) => {
 const response = message as INotificationOperationResponse;
 //custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext('browserTabId','msiTabId');
 let notificationContext = await tabContext.getNotificationContext('notificationId007');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('CloseNotification');
 notificationContext.publish(requestObject).then((resp) => {
 //custom code
 }).catch((err) => {
 //custom code
 });
```

Listen to close event of the notification

Here's a Typescript example to listen to the close notification. Below code contains code to listen close action of already opened notification with id notificationId007:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent') as
 IEventRequest;
notificationContext.subscribe(requestObject, (response) => {
 const resp: INotificationCloseActionEventResponse = response as INotificationCloseActionEventResponse;
 console.log(resp.getResponseData().getNotificationId())
});
```

Here's a Typescript example. The first part is to open a notification with id, notificationId007. Once it's opened and the result succeeds, we can add the code to listen to the close notification:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
requestObject.setTitle('title');
requestObject.setSummary('Error message summary');
notificationContext.publish(requestObject).then((message) => {
 const response: INotificationOperationResponse = message as INotificationOperationResponse;
 //given below is the code to listen to close notification of this notification

 const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent') as
 IEventRequest;
 notificationContext.subscribe(requestObject, (response) => {
 const resp: INotificationCloseActionEventResponse = response as INotificationCloseActionEventResponse;
 console.log(resp.getResponseData().getNotificationId())
 });

});
```

Here's a JavaScript example of listening to the close notification. The following code shows an example of listening to a close action of an already opened notification with id notificationId007:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent');
 notificationContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId())
 });
```

Here's a JavaScript example of listening to the close notification. The first part is to open a notification with id, notificationId007. Once it's opened and the result is success, we can add the code to listen to the close notification to listen to the close notification. Below code contains code to listen close action of already opened notification with id notificationId007:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle('title');
notificationContext.publish(requestObject).then((message) => {
 //given below is the code to listen to close notification of this notification
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent');
 notificationContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId());
 });
});
```

Listen to actions given in the notification

Here's a Typescript example to listen to the action triggered in the notification. Below code contains code to listen action of already opened notification with id notificationId007:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationActionEvent') as
 IEventRequest;
 notificationContext.subscribe(requestObject, (response) => {
 const resp: INotificationActionEventResponse = response as INotificationActionEventResponse;
 console.log(resp.getResponseData().getNotificationId())
 console.log(resp.getResponseData().getActionId())
 console.log(resp.getResponseData().getActionName())
 });
```

Here's a Typescript. The first part is to open a notification with id, notificationId007. Once it's opened and the result is success, we can add the code to listen actions triggered in the notification. In this example, 2 actions are added, Join call and reject call. These two actions will show as buttons in the notification. Once any of this buttons clicked, can listen to that particular action triggered with details including action id and action name. :

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
const requestObject: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest;
requestObject.setTitle('title');
requestObject.setActions([{id: 'join', name: 'Join Call'}, {id: 'reject', name: 'Reject Call'}]);
notificationContext.publish(requestObject).then((message) => {
 const response: INotificationOperationResponse = message as INotificationOperationResponse;
 //given below is the code to listen to actions triggered in the notification of this notification
 const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationActionEvent') as
 IEventRequest;
 notificationContext.subscribe(requestObject, (response) => {
 const resp: INotificationCloseActionEventResponse = response as INotificationCloseActionEventResponse;
 console.log(resp.getResponseData().getNotificationId());
 console.log(resp.getResponseData().getActionId());
 console.log(resp.getResponseData().getActionName());
 });
});
```

Here's a JavaScript example to listen to the action triggered in the notification. Below code contains code to listen action of already opened notification with id notificationId007:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationActionEvent');
 notificationContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId())
 console.log(response.getResponseData().getActionId())
 console.log(response.getResponseData().getActionName())
});
```

Here's a JavaScript example to listen to to open a notification with id, notificationId007. Once it's opened and the result is success, we can add the code to listen actions triggered in the notification. In this example, 2 actions are added, Join call and reject call. These two actions will show as buttons in the notification. Once any of this buttons clicked, we can listen to that particular action triggered with details including action id and action name. :

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
requestObject.setTitle('title');
requestObject.setActions([{id: 'join', name: 'Join Call'}, {id: 'reject', name: 'Reject Call'}]);
notificationContext.publish(requestObject).then((message) => {
 //given below is the code to listen to actions triggered in the notification of this notification
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationActionEvent');
 notificationContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId());
 console.log(response.getResponseData().getActionId());
 console.log(response.getResponseData().getActionName());
 });
});
```

UI Events Framework and MCA Events

onToolbarAgentCommand

This event subscription should be added as part of initialization of MCA integration in the CTI application.

This event is used to pass interaction commands to the application from CTI. This is a controllable event. So, if a user wants to control this event by either resolving or rejecting it, they must pass a promise as the return of this event callback function. On notification of this event subscription, user can check the command (for example, getActiveInteractionCommands) in the event response and pass outData for that command by resolving the promise with that outData. The event subscription should be added only once, during initialization of the MCA code for CTI integrations.

Note: The user will get this event notification only once for a session.

Here's a Typescript example for adding subscription for onToolbarAgentCommand event.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarAgentCommand') as
 IMcaEventRequest;
 phoneContext.subscribe(request, (response: IEventResponse) => {
 const dataUpdateResponse = response as IMcaOnToolbarAgentCommandEventResponse;
 const data: IMcaOnToolbarAgentCommandDataResponse = dataUpdateResponse.getResponseData();
 const command: string = data.getCommand();
 return new Promise((resolve, reject) => {
 switch (command) {
 case "makeAvailable":
 break;
 case "makeUnavailable":
 break;
```

```
case "getActiveInteractionCommands":
const outData = {
'channel': 'PHONE',
'supportedCommands': [{
'name': "hold", 'toggleCommand': "unhold", 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Hold", 'position': 1, 'icon': ""
},
{
'name': "unhold", 'toggleCommand': "hold", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Off Hold", 'position': 1, 'icon': ""
},
{
'name': "mute", 'toggleCommand': "unmute", 'defaultEnabled': true,
'defaultVisible': true, 'displayeName': "Mute", 'position': 2, 'icon': ""
},
{
'name': "unmute", 'toggleCommand': "mute", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Unmute", 'position': 2, 'icon': ""
},
{
'name': "record", 'toggleCommand': "stopRecord", 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Record", 'position': 3, 'icon': ""
},
{
'name': "stopRecord", 'toggleCommand': "record", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Stop Recording", 'position': 3, 'icon': ""
},
{
'name': "transfer", 'toggleCommand': null, 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Transfer", 'position': 4, 'icon': ""
},
{
'name': "disconnect", 'toggleCommand': null, 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Hang Up", 'position': 5, 'icon': ""
}]
};
data.setOutdata(outData);
break;
}
data.setResult('success');
resolve(data);
});
});
```

Here's a JavaScript example for adding subscription for onToolbarAgentCommand event.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarAgentCommand');
phoneContext.subscribe(request, (response) => {
 const dataUpdateResponse = response;
 const data = dataUpdateResponse.getResponseData();
 const command = data.getCommand();
 return new Promise((resolve, reject) => {
 switch (command) {
 case "makeAvailable":
 break;
 case "makeUnavailable":
 break;
 case "getActiveInteractionCommands":
 const outData = {
 'channel': 'PHONE',
 'supportedCommands': [{
```

```
'name': "hold", 'toggleCommand': "unhold", 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Hold", 'position': 1, 'icon': ""
},
{
'name': "unhold", 'toggleCommand': "hold", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Off Hold", 'position': 1, 'icon': ""
},
{
'name': "mute", 'toggleCommand': "unmute", 'defaultEnabled': true,
'defaultVisible': true, 'displayeName': "Mute", 'position': 2, 'icon': ""
},
{
'name': "unmute", 'toggleCommand': "mute", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Unmute", 'position': 2, 'icon': ""
},
{
'name': "record", 'toggleCommand': "stopRecord", 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Record", 'position': 3, 'icon': ""
},
{
'name': "stopRecord", 'toggleCommand': "record", 'defaultEnabled': false,
'defaultVisible': false, 'displayName': "Stop Recording", 'position': 3, 'icon': ""
},
{
'name': "transfer", 'toggleCommand': null, 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Transfer", 'position': 4, 'icon': ""
},
{
'name': "disconnect", 'toggleCommand': null, 'defaultEnabled': true,
'defaultVisible': true, 'displayName': "Hang Up", 'position': 5, 'icon': ""
}]
};
data.setOutdata(outData);
break;
}
data.setResult('success');
resolve(data);
});
});
```

onToolbarInteractionCommand

Once a call is received at the Fusion application, the Fusion application can accept, reject or disconnect an active call.

The onToolbarInteractionCommand is fired from the Fusion application with this information (accept, reject or disconnect). This is a controllable event. So, if the user wants to control this event by either resolving or rejecting it, they must pass a promise as the return of this event callback function. On notification of this event subscription, user can check the command in the event response. Commands supported are, accept, disconnect, reject and setActive.

These commands corresponds to accept disconnect reject operations agent might make in the Fusion application. User can wire up their logic for each of these commands, on the callback of this event response. The event subscription should be added only once, during initialization of the MCA code for CTI integrations.

Example Use Case

When a CTI supplier makes a call request, we should start a NewCommAction Request to Fusion application, using UEF publish api. This will show a call notification in Fusion application to either accept or decline the call. Now, Agent can either accept this call or decline this call. According to the mentioned action from agent, the CTI integration should

either call api to make the call connection for call accept action by agent or, the CTI integration should reject the entire call request for decline action by agent.

In this scenario, onToolbarAgentCommand will give a notification to the CTI integration application about the choice agent has made, ie, either accept or decline and based on that CTI integration can wire up the logic to either call UEF StartComm Api to enable call connection, or call UEF CloseComm API to reject the call entirely.

Here's a Typescript example for adding subscription for onToolbarInteractionCommand event.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand') as
 IMcaEventRequest;
 phoneContext.subscribe(request, (response: IEventResponse) => {
 const dataUpdateResponse = response as IMcaOnToolbarInteractionCommandEventResponse;
 const data: IMcaOnToolbarInteractionCommandDataResponse = dataUpdateResponse.getResponseData();
 const command: string = data.getCommand();
 return new Promise((resolve, reject) => {
 switch (command) {
 case "accept":
 // publish startCommEvent action
 break;
 case "disconnect":
 // publish closeCommEvent with proper reason
 break;
 case "reject":
 // publish closeCommEvent with proper reason
 break;
 }
 data.setResult('success');
 resolve(data);
 });
 });
```

Here's a JavaScript example adding subscription for onToolbarInteractionCommand event.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand');
phoneContext.subscribe(request, (response) => {
 const dataUpdateResponse = response;
 const data = dataUpdateResponse.getResponseData();
 const command = data.getCommand();
 return new Promise((resolve, reject) => {
 switch (command) {
 case "accept":
 // publish startCommEvent action
 break;
 case "disconnect":
 // publish closeCommEvent with proper reason
 break;
 case "reject":
 // publish closeCommEvent with proper reason
 break;
 }
```

```
 data.setResult('success');
 resolve(data);
 });
});
```

onDataUpdated

This event is used to listen to a DataUpdate event that happens in Fusion application.

This is a non controllable event. Using this listener, the Fusion application (Service Center) can send notifications about updates of Fusion application (Service Center) such as contact information which will then be used to show in the CTI toolbar. Also, the Fusion application can use this method to notify the toolbar about the Fusion application actions, such as a Wrap-up, call have been completed and so on.

Here's a Typescript example for adding subscription for onDataUpdated event.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const request: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated') as IMcaEventRequest;
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer to Application Classification Code for the list of supported app classifications
request.setAppClassification('_appClassfication');
phoneContext.subscribe(request, (response: IEventResponse) => {
 const dataUpdateResponse = response as IMcaOnDataUpdatedEventResponse;
 const data: IMcaOnDataUpdatedData = dataUpdateResponse.getResponseData();
 // Custom Code
});
```

Here's a JavaScript example adding subscription for adding subscription for onDataUpdated event.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer to Application Classification Code for the list of supported app classifications
request.setAppClassification('_appClassfication');
phoneContext.subscribe(request, (response) => {
 const dataUpdateResponse = response;
 const data = dataUpdateResponse.getResponseData();
 // Custom Code
});
```

onOutgoingEvent

This event is used to listen to an outgoing communication event that generates from Fusion application application.

This is a non controllable event. Using this listener CTI can wire up the logic to start a call, for example, to call the UEF NewComm API, based on the information in the event response of this event.

Example Use Case

Agent clicking on a Contact mobile number link from Fusion application application from Service Request Edit page, or agent clicking a mobile link of a Contact from Digital Sales Contact page. These actions will trigger the OnOutgoingEvent and the CTI application will get the notification if it has added a subscription for this event.

Here's a Typescript example for adding subscription for onOutgoingEvent event.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onOutgoingEvent') as IMcaEventRequest;
 request.setAppClassification('_appClassfication');
 phoneContext.subscribe(request, (response: IEventResponse) => {
 const dataUpdateResponse = response as IMcaonOutgoingEventResponse;
 const data: IMcaOnOutgoingEventData = dataUpdateResponse.getResponseData();
 // Custom Code
 });
```

Here's a JavaScript example adding subscription for adding subscription for onOutgoingEvent event.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onOutgoingEvent');
request.setAppClassification('_appClassfication');
phoneContext.subscribe(request, (response) => {
 const dataUpdateResponse = response;
 const data = dataUpdateResponse.getResponseData();
 // Custom Code
});
```

UI Events Framework and MCA Operations

getConfiguration

This operation is to get configuration information that enables the toolbar to evaluate the features supported by Fusion application.

Based on it, the toolbar informs Fusion application of any features that must be disabled. This is a required API that should be called during initialization of MCA integration code in CTI application, with which client application can get the currently logged in agent details, supported features, and so on.

Here's a Typescript example calling the getConfiguration operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const requestObject: IMcaGetConfigurationActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('getConfigurationOperation') as
 IMcaGetConfigurationActionRequest;
requestObject.setConfigType("TOOLBAR");
phoneContext.publish(requestObject).then((operationResponse) => {
 const mcaGetConfigurationActionResponseData: IMcaGetConfigurationActionResponseData = (operationResponse as
 IMcaGetConfigurationActionResponse).getResponseData();
 const isSuccess = mcaGetConfigurationActionResponseData.isSuccess();
 const mcaGetConfiguration = mcaGetConfigurationActionResponseData.getConfiguration();
 const AgentId = mcaGetConfiguration.getAgentId();
 const AgentPartyId = mcaGetConfiguration.getAgentPartyId();
 const Features = mcaGetConfiguration.getFeatures();
 const CompanionPanelUrl = mcaGetConfiguration.getCompanionPanelUrl();
 const CompanionPanelTitle = mcaGetConfiguration.getCompanionPanelTitle();
 const FaTrustToken = mcaGetConfiguration.getFaTrustToken();
 console.log('mcaGetConfigurationActionResponseData', mcaGetConfigurationActionResponseData);
 console.log('isSuccess', isSuccess);
 console.log('mcaGetConfiguration', mcaGetConfiguration);
 console.log('AgentId', AgentId);
 console.log('AgentPartyId', AgentPartyId);
 console.log('Features', Features);
 console.log('CompanionPanelUrl', CompanionPanelUrl);
 console.log('CompanionPanelTitle', CompanionPanelTitle);
 console.log('FaTrustToken', FaTrustToken);
}).catch((error) => console.log(error));
```

Here's a JavaScript example of calling the getConfiguration operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const requestObject =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('getConfigurationOperation');
 requestObject.setConfigType("TOOLBAR");
 this.phoneContext.publish(requestObject).then((operationResponse) => {
 const mcaGetConfigurationActionResponseData = operationResponse.getResponseData();
 const isSuccess = mcaGetConfigurationActionResponseData.isSuccess();
 const mcaGetConfiguration = mcaGetConfigurationActionResponseData.getConfiguration();
 const AgentId = mcaGetConfiguration.getAgentId();
 const AgentPartyId = mcaGetConfiguration.getAgentPartyId();
 const Features = mcaGetConfiguration.getFeatures();
 const CompanionPanelUrl = mcaGetConfiguration.getCompanionPanelUrl();
 const CompanionPanelTitle = mcaGetConfiguration.getCompanionPanelTitle();
 const FaTrustToken = mcaGetConfiguration.getFaTrustToken();
 console.log('mcaGetConfigurationActionResponseData', mcaGetConfigurationActionResponseData);
 console.log('isSuccess', isSuccess);
 console.log('mcaGetConfiguration', mcaGetConfiguration);
 console.log('AgentId', AgentId);
 console.log('AgentPartyId', AgentPartyId);
 console.log('Features', Features);
 console.log('CompanionPanelUrl', CompanionPanelUrl);
 console.log('CompanionPanelTitle', CompanionPanelTitle);
 console.log('FaTrustToken', FaTrustToken);
 }).catch((error) => console.log(error));
```

agentStateEvent

This operation notifies the Fusion application of a change in the user's signed in or availability status for the specified channel. This API should be called immediately after the initialization to make agent available for calls, even though the operation name includes "event".

Here's a Typescript example calling the agentStateEvent operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const requestObject: IMcaAgentStateEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('agentStateEventOperation') as
 IMcaAgentStateEventActionRequest;
const state: boolean = true;
 requestObject.setEventId('1');
 requestObject.setIsAvailable(state); /* true or false */
 requestObject.setIsLoggedIn(true); /* true or false */
 requestObject.setState(state ? 'AVAILABLE' : 'UNAVAILABLE'); /*'AVAILABLE' : 'UNAVAILABLE'*/
 requestObject.setStateDisplayString(state ? 'Idle' : 'On Break. Lunch break'); /* 'Idle' : 'On Break. Lunch
 break'*/
 requestObject.setReason(null as any);
 requestObject.setReasonDisplayString(state ? 'Idle' : 'On Break'); /* 'Idle' : 'On Break' */
 requestObject.setInData({ 'phoneLineId': '1' }); /* { 'phoneLineId': '1' } */
 phoneContext.publish(requestObject).then((operationResponse: IOperationResponse) => {
 const mcaAgentStateEventActionResponseData: IMcaAgentStateEventActionResponseData = (operationResponse as
 IMcaAgentStateEventActionResponse).getResponseData();
 mcaAgentStateEventActionResponseData.isSuccess()
 }).catch((error) => {
 console.log(error);
 });
```

Here's a JavaScript example calling the agentStateEvent operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const requestObject =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('agentStateEventOperation');
const state = true;
 requestObject.setEventId('1');
 requestObject.setIsAvailable(state); /* true or false */
 requestObject.setIsLoggedIn(true); /* true or false */
 requestObject.setState(state ? 'AVAILABLE' : 'UNAVAILABLE'); /*'AVAILABLE' : 'UNAVAILABLE'*/
 requestObject.setStateDisplayString(state ? 'Idle' : 'On Break. Lunch break'); /* 'Idle' : 'On Break. Lunch
 break'*/
 requestObject.setReason(null);
 requestObject.setReasonDisplayString(state ? 'Idle' : 'On Break'); /* 'Idle' : 'On Break' */
 requestObject.setInData({ 'phoneLineId': '1' }); /* { 'phoneLineId': '1' } */
 phoneContext.publish(requestObject).then((operationResponse) => {
 const mcaAgentStateEventActionResponseData = operationResponse.getResponseData();
 mcaAgentStateEventActionResponseData.isSuccess()
 }).catch((error) => {
 console.log(error);
```

```
});
```

disableFeature

This operation notifies the Fusion application that a subset of available functionality must be disabled because the toolbar hasn't implemented it.

Here's a Typescript example calling the disableFeature operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const request: IMcaDisableFeatureActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('disableFeatureOperation') as
 IMcaDisableFeatureActionRequest;
 request.setFeatures(['OUTGOING', 'INCOMING']);
 phoneContext.publish(request).then((operationResponse) => {
 const mcaGetConfigurationActionResponseData: IMcaDisableFeatureActionResponseData = (operationResponse as
 IMcaDisableFeatureActionResponse).getResponseData();
 });
```

Here's a JavaScript example calling the disableFeature operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('disableFeatureOperation');
 request.setFeatures(['OUTGOING', 'INCOMING']);
 phoneContext.publish(request).then((operationResponse) => {
 const mcaGetConfigurationActionResponseData = operationResponse.getResponseData();
 });
```

readyForOperation

This operation notifies the Fusion application that toolbar is ready for operation.

Here's a Typescript example calling the readyForOperation operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const request: IMcaReadyForOperationActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('readyForOperationOperation') as
 IMcaReadyForOperationActionRequest;
request.setReadiness(true);
phoneContext.publish(request).then((operationResponse) => {
```

```
 const mcaGetConfigurationActionResponseData: IMcaReadyForOperationActionResponseData = (operationResponse
 as IMcaReadyForOperationActionResponse).getResponseData();
 });
```

Here's a JavaScript example calling the readyForOperation operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('readyForOperationOperation');
request.setReadiness(true);
phoneContext.publish(request).then((operationResponse) => {
 const mcaGetConfigurationActionResponseData = operationResponse.getResponseData();
});
```

newCommEvent

This is the action user needs to publish when there's a ring received scenario.

For example, when a user is trying to call an agent, the Fusion application application should show a notification with Accept or Decline Button. To enable this the MCA integrated CTI should publish a newCommEvent request.

> **Note:** This API is called to try for a call connection, based on agents accept or decline actions, call will be established or completely declined with StartCommEvent and CloseCommEvent respectively.

Here's a Typescript example calling the newCommEvent operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent') as
 IMcaNewCommEventActionRequest;
 request.setAppClassification('appClassfication'); // You should set correct appClassification here. eg:
 ORA_SERVICE for Service App
    request.setEventId('1'); // Set the event ID here
 // request.setInputData(_inboundData); _inboundData of type - IMcaNewCommInData
 const inData: IMcaInDataRequest = request.getInData();
 request.getInData().setCallStatus('INCOMING'); // optional for incoming calls
    request.getInData().setInDataValueByAttribute('SVCMCA_ANI', '+12345678');
    phoneContext.publish(request).then((res: IOperationResponse) => {
 const response: IMcaNewComActionResponse = res as IMcaNewComActionResponse;
 const mcaNewComActionData: IMcaNewComActionData = response.getResponseData();
 const mcaOutData: IMcaOutData = mcaNewComActionData.getOutData(); // This outData should be passed as
 inData to startCommEvent or closeCommEvent based on the scenario about call connection
 }).catch(() => {
 })
```

Here's a JavaScript example calling the newCommEvent operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
```

```
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('appClassfication');
request.setEventId('1'); // Set the event ID here
// request.setInputData(_inboundData); _inboundData of type - IMcaNewCommInData
const inData = request.getInData();
request.getInData().setCallStatus('INCOMING'); // optional for incoming calls

request.getInData().setInDataValueByAttribute('SVCMCA_ANI', '+12345678');
phoneContext.publish(request).then((res) => {
 const mcaNewComActionData = res.getResponseData();
 const mcaOutData = mcaNewComActionData.getOutData(); // This outData should be passed as inData to
 startCommEvent or closeCommEvent based on the scenario about call connection
}).catch(() => {

})
```

startCommEvent

This is the action the user must publish when there's a call-accepted scenario and the real call connection needs to be established

For instance, when an agent clicks Accept on a call notification, the CTI integration code base would get notification for onAgentInteractionCommand Event with command as Accept, if it has a subscription added for the same.

Once it gets a notification for the call accept command, the CTI integration should publish a startCommEvent operation, which enables the call connection and also loads the interaction component for this call in UI. For outgoing event also, when the CTI gets an event notification for onOutgoingEvent, it needs to call the startCommEvent publish API to establish a outgoing call.

> **Note:** This API is called for establishing the call connection. The inData parameter that must be passed for StartCommEvent operation request should be matching with the outData response we get in the newCommEvent operation response. Similarly the inData for StartCommEvent operation request, for an OutGoingCall event from fusion application must be the response for the OutGoingCallEvent notification.

Here's a Typescript example calling the startCommEvent operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
    // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
    request.setAppClassification('appClassfication');
    request.setEventId('1'); // Set the event ID here
 // request.setInputData(_inboundData);// _inboundData of type - IMcaStartCommInData
```

```
const inData: IMcaStartCommInDataRequest = request.getInData();
phoneContext.publish(request).then((res: IOperationResponse) => {
const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
}).catch(() => {
})
```

Here's a JavaScript example calling the startCommEvent operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('appClassfication');
// request.setInputData(_inboundData);// _inboundData of type - IMcaStartCommInData
const inData = request.getInData();
phoneContext.publish(request).then((res) => {
}).catch(() => {
})
```

EngagementContext to interact with UI Events Framework integrated objects

Here's a Typescript example for startCommEvent using UEF for subscribing to context open events from the tab context retrieved from the engagement context. From this tab context, active record can be retrieved using tabContext.getActiveRecord() method.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');

const request: IMcaStartCommEventActionRequest =
uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
IMcaStartCommEventActionRequest;
   // You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('self.appClassfication');
   request.setEventId('1'); // Set the event ID here
   request.getInData().setInDataValueByAttribute("SVCMCA_SR_NUM", "SR00001"); // Pass a valid SR number
here, so that this R will gets opened when startComm is fired
const response: IMcaStartComActionResponse = await phoneContext.publish(request) as
IMcaStartComActionResponse;
const contactId: string =
response.getResponseData().getEngagementContext().getEngagementData().getOutputData().getSVCMCA_CONTACT_NUMBER();

       // get tab context from startComm event's engagementContext.
       const engagementContext: IEngagementContext = response.getResponseData().getEngagementContext();
const tabContext: ITabContext = await engagementContext.getTabContext();

       // listen to context open event from the tabContext retrieved from engagementContext
       const coPayload =
uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent') as
IFieldValueChangeEventRequest;
       tabContext.subscribe(coPayload, (coRes) => {
const coResponse = coRes as IContextOpenEventResponse;
console.log("Object Type: ", coResponse.getResponseData().getRecordType());
   console.log("Object ID: ", coResponse.getResponseData().getRecordId());
       });
```

Here's a JavaScript example for startCommEvent using UEF for subscribing to context open events from the tab context retrieved from the engagement context. From this tab context, active record can be retrieved using tabContext.getActiveRecord() method.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer to Application Classification Code for the list of supported app classifications
request.setAppClassification('self.appClassfication');
request.setEventId('1'); // Set the event ID here
request.getInData().setInDataValueByAttribute("SVCMCA_SR_NUM", "SR00001"); // Pass a valid SR number here,
 so that this R will gets opened when startComm is fired
const response = await phoneContext.publish(request);
const contactId =
 response.getResponseData().getEngagementContext().getEngagementData().getOutputData().getSVCMCA_CONTACT_NUMBER();

// get tab context from startComm event's engagementContext.
const engagementContext = response.getResponseData().getEngagementContext();
const tabContext = await engagementContext.getTabContext();

// listen to context open event from the tabContext retrieved from engagementContext
const coPayload =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
tabContext.subscribe(coPayload, (coRes) => {
 const coResponse = coRes;
 console.log("Object Type: ", coResponse.getResponseData().getRecordType());
 console.log("Object ID: ", coResponse.getResponseData().getRecordId());
});
```

closeCommEvent

For this action the user must publish when there's a call-decline scenario.

For instance, when an agent clicks Decline on a call notification, CTI integration code base would get notification for onAgentInteractionCommand Event with command as 'decline', if it has a subscription added for the same (Please see the onAgentInteractionCommand example for the same).

Now, once it gets a notification for the call decline command, the CTI integration should publish a closeCommEvent operation, which disconnects the entire call For outgoing event also, when the CTI gets an event notification for onOutgoingEvent and if it wants to reject that call, it needs to call the closeCommEvent publish API.

> **Note:** The inData parameter that needs to be passed for closeCommEvent operation request should be matching with the outData response we get in the newCommEvent operation response. The setReason API in closeComm inData request will enable user to set the reason for the closeCommEvent operation. (for example, reason as MISSED, REJECT, WRAPUP and so on.

Here's a Typescript example calling the closeCommEvent operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

```
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request: IMcaCloseCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent') as
 IMcaCloseCommEventActionRequest;
    // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
    request.setAppClassification('appClassfication');
    request.setEventId('1'); // Set the event ID here
 // request.setInputData(_inboundData);// _inboundData of type - IMcaStartCommInData
 request.setReason('WRAPUP'); // Specify the reason
 const inData = request.getInData();
 phoneContext.publish(request).then((res: IOperationResponse) => {
 const response = res as IMcaCloseComActionResponse
 }).catch(() => {
 })
```

Here's a JavaScript example calling the closeCommEvent operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent');
// You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('appClassfication');
request.setEventId('1'); // Set the event ID here
// request.setInputData(_inboundData);// _inboundData of type - IMcaStartCommInData
request.setReason('WRAPUP'); // Specify the reason
const inData = request.getInData();
phoneContext.publish(request).then((res) => {
}).catch(() => {
})
```

outboundCommError

The toolbar can publish this API to notify the fusion application that an error occurred during initiation of the outbound call. The error occurs if the identifier of the event, such as phone number, or email can't be used to establish a connection. In this case user needs to call outboundCommError API with event id as same as the outbound call, to reject it.

Here's a Typescript example calling the outboundCommError operation.

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request: IMcaOutboundCommErrorActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('outboundCommError') as
 IMcaOutboundCommErrorActionRequest;
 request.setCommUuid('eventId'); // set the event id here
 request.setErrorCode('INVALID_NUMBER');
 request.setErrorMsg('{"phoneLineId":phoneLineIdString}');
 phoneContext.publish(request).then((operationResponse: IOperationResponse) => {
```

```
 console.log('outboundCommError', operationResponse);
 const mcaOutBoundCommErrorResponsePayload: IMCAOutBoundCommErrorResponsePayload = (operationResponse as
 IMcaOutBoundCommErrorActionResponse).getResponseData().getData()
 }).catch((error: IErrorData) => {
 // console.log(error);
 });
```

Here's a JavaScript example calling the outboundCommError operation.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('outboundCommError');
request.setCommUuid('eventId'); // set the event id here
request.setErrorCode('INVALID_NUMBER');
request.setErrorMsg('{"phoneLineId":phoneLineIdString}');
phoneContext.publish(request).then((operationResponse) => {
 const mcaOutBoundCommErrorResponsePayload = (operationResponse).getResponseData().getData()
}).catch((error) => {
 // console.log(error);
});
```

InteractionLogging

Interaction Logging

startLogging

Here's a Typescript example to start logging:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
```

Here's a JavaScript example to start logging:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
```

Here's a Typescript example to start logging with an extensible field:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging({'UefCustomTextTvm_c':'Extensible
 field value updated!'});
```

Here's a JavaScript example to start logging with an extensible field:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging({'UefCustomTextTvm_c':'Extensible field value
 updated!'});
```

stopLogging

Here's a Typescript example to stop logging an interaction:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
interactionLogger.stopLogging();
```

Here's a JavaScript example to stop logging an interaction

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
interactionLogger.stopLogging();
```

Here's a Typescript example to stop logging an interaction by updating an extensible field:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
interactionLogger.stopLogging({'UefCustomTextTvm_c':'Extensible field value updated!'},
 {'InteractionNotes':'quick 123'});
```

Here's a JavaScript example to stop logging an interaction by updating an extensible field:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
interactionLogger.stopLogging({'UefCustomTextTvm_c':'Extensible field value updated!'});
```

\

Here's a Typescript example to stopLogging and then create a wrap-up object:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
```

Here's a JavaScript example to stopLogging and then create a wrap-up object:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
interactionLogger.stopLogging(null, {'InteractionNotes':'quick 123'});
```

pauseLogging

Here's a Typescript example to pause logging an interaction

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
```

Here's a JavaScript example to pause logging an interaction:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
```

resumeLogging

Here's a Typescript example to resume logging an interaction:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
await interactionLogger.resumeLogging();
```

Here's a JavaScript example to resume logging an interaction:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
await interactionLogger.resumeLogging();
```

Insights

Publish Action to show Insights

Here's a Typescript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
insightContext.publish(payload).then((response: IOperationResponse) => {
```

```
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 let recordContext = await tabContext.getActiveRecord();
 let insightContext = await recordContext.getInsightsContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('ShowInsights');
 payload.setId('insightsId1');
 payload.setTitle('Custom title');
 insightContext.publish(payload).then((response) => {
 const insightsOperationResponse = response;
 console.log(insightsOperationResponse.getResponseData().getInsightsId());
 });
```

Publish Action to show Insights with message

Here's a Typescript example to show Insights, considering recordContext as current browser tabs active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
payload.setMessage('Custom message to show details about the suggestion');
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example to show Insights, considering recordContext as current browser tabs active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
payload.setHeader('Custom category');
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
});
```

Publish Action to show Insights with Category

Here's a Typescript example to show Insights, considering recordContext as current browser tabs active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
payload.setHeader('Custom category');
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example to show Insights, considering recordContext as current browser tabs active record

```
 const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 let recordContext = await tabContext.getActiveRecord();
 let insightContext = await recordContext.getInsightsContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('ShowInsights');
 payload.setId('insightsId1');
 payload.setTitle('Custom title');
 payload.setHeader('Custom category');
 insightContext.publish(payload).then((response) => {
 const insightsOperationResponse = response;
 console.log(insightsOperationResponse.getResponseData().getInsightsId());
 });
```

Publish Action to show Insights with Custom Icon

Here's a Typescript example to show Insights, considering r recordContext as current browser tabs active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
payload.setIcon('oj-ux-ico-blog-post-detail');
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
```

```
});
```

Here's a JavaScript example to show Insights, considering recordContext as current browser tabs active record

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 let recordContext = await tabContext.getActiveRecord();
 let insightContext = await recordContext.getInsightsContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('ShowInsights');
 payload.setId('insightsId1');
 payload.setTitle('Custom title');
 payload.setIcon('oj-ux-ico-blog-post-detail');
 insightContext.publish(payload).then((response) => {
 const insightsOperationResponse = response;
 console.log(insightsOperationResponse.getResponseData().getInsightsId());
 });
```

Publish Action to show Insights with Custom Action

Here's a Typescript example to show Insights, considering recordContext as current browser tabs active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID','V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
 as IShowInsightsRequest;
payload.setId('insightsId1');
payload.setTitle('Custom title');
(payload.action() as IInsightsActionRequest).setActionDetails('Action Name','actionId');
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsOperationResponse = response as IInsightsOperationResponse;
 console.log((insightsOperationResponse.getResponseData() as
 IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example to show Insights, considering recordContext as current browser tabs active record:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 let recordContext = await tabContext.getActiveRecord();
 let insightContext = await recordContext.getInsightsContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('ShowInsights');
 payload.setId('insightsId1');
 payload.setTitle('Custom title');
 payload.action().setActionDetails('Action Name','actionId');
 insightContext.publish(payload).then((response) => {
 const insightsOperationResponse = response;
 console.log(insightsOperationResponse.getResponseData().getInsightsId());
 });
```

Publish Action to show multiple insights from an external source

Here's a Typescript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
let dataFromExternalSource = [
{
'id': 'insights1',
'title': 'custom title 1',
'message': 'custom message 1',
'header': 'category 1',
'actionName': 'Action 1',
'actionId': 'act1'
},
{
'id': 'insights2',
'title': 'custom title 2',
'message': 'custom message 2',
'header': 'category 2',
'actionName': 'Action 2',
'actionId': 'act2',
}
];

dataFromExternalSource.forEach((element: any) => {
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('ShowInsights')
as IShowInsightsRequest;
payload.setId(element.id);
payload.setTitle(element.title);
payload.setMessage(element.message);
payload.setHeader(element.header);
(payload.action() as IInsightsActionRequest).setActionDetails(element.actionName, element.actionId);
insightContext.publish(payload).then((response: IOperationResponse) => {
const insightsOperationResponse = response as IInsightsOperationResponse;
console.log((insightsOperationResponse.getResponseData() as
IInsightsOperationResponseData).getInsightsId());
});
});
```

Here's a JavaScript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext = await tabContext.getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
let dataFromExternalSource = [
{
'id': 'insights1',
'title': 'custom title 1',
'message': 'custom message 1',
'header': 'category 1',
'actionName': 'Action 1',
'actionId': 'act1'
},
{
```

```
'id': 'insights2',
'title': 'custom title 2',
'message': 'custom message 2',
'header': 'category 2',
'actionName': 'Action 2',
'actionId': 'act2',
}
];
dataFromExternalSource.forEach((element) => {
const payload = frameworkProvider.requestHelper.createPublishRequest('ShowInsights');
payload.setId(element.id);
payload.setTitle(element.title);
payload.setMessage(element.message);
payload.setHeader(element.header);
payload.action().setActionDetails(element.actionName, element.actionId);
insightContext.publish(payload).then((response) => {
const insightsOperationResponse = response;
console.log(insightsOperationResponse.getResponseData().getInsightsId());
});
});
```

Publish Action to dismiss Insights

Here's a Typescript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext()
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest =
frameworkProvider.requestHelper.createPublishRequest('DismissInsights') as IShowInsightsRequest;
payload.setId('insightsId1');
insightContext.publish(payload).then((response: IOperationResponse) => {
const insightsOperationResponse = response as IInsightsOperationResponse;
console.log((insightsOperationResponse.getResponseData() as
IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext = await tabContext.getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('DismissInsights');
payload.setId('insightsId1');
insightContext.publish(payload).then((response) => {
const insightsOperationResponse = response;
console.log(insightsOperationResponse.getResponseData().getInsightsId());
});
```

Publish Action to get all Insights

Here's a Typescript example to show Insights, considering recordContext as current the current browser tab's active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext()
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest = frameworkProvider.requestHelper.createPublishRequest('GetAllInsights')
 as IShowInsightsRequest;
insightContext.publish(payload).then((response: IOperationResponse) => {
 const insightsGetAllOperationResponse = response as IInsightsGetAllOperationResponse;
 console.log((insightsGetAllOperationResponse.getResponseData() as
 IInsightsGetAllOperationResponseData).getInsights());
});
```

Here's a JavaScript example to show Insights, considering recordContext as the current browser tab's active record:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext = await tabContext.getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('GetAllInsights');
insightContext.publish(payload).then((response) => {
 const insightsGetAllOperationResponse = response;
 console.log(insightsGetAllOperationResponse.getResponseData().getInsights());
});
```

Listen Insights dismiss event with reason

In the close response, you can get the reason of close from getReason function. It gives 3 reasons in response:

1. PROGRAMMATIC if the insights closed by dismiss insights action from UEF
2. MANUAL if the notification closed by clicking the dismiss link in the insights.
3. ACTION if the notification closed by clicking the action given.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext()
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IInsightsSubscriptionRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent') as
 IInsightsSubscriptionRequest;
payload.setId('insightsId1');
insightContext.subscribe(payload, (response: IEventResponse ) => {
 const insightsDismissActionEventResponse = response as IInsightsDismissActionEventResponse;
 console.log((insightsDismissActionEventResponse.getResponseData() as
 IInsightsDismissActionData).getInsightsId());
 console.log((insightsDismissActionEventResponse.getResponseData() as
 IInsightsDismissActionData).getReason());
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
 let recordContext = await tabContext.getActiveRecord();
 let insightContext = await recordContext.getInsightsContext();
 const payload =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent');
 payload.setId('insightsId1');
 insightContext.subscribe(payload, (response) => {
 const insightsDismissActionEventResponse = response;
 console.log(insightsDismissActionEventResponse.getResponseData().getInsightsId());
 console.log(insightsDismissActionEventResponse.getResponseData().getReason());
 });
```

Listen Insights Action executing event

Here's a Typescript example considering recordContext as the current browser tab's active record:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext: IRecordContext = await tabContext.getActiveRecord();
let insightContext: IInsightsContext = await recordContext.getInsightsContext();
const payload: IInsightsSubscriptionRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsActionEvent') as
 IInsightsSubscriptionRequest;
payload.setId('insightsId1');
insightContext.subscribe(payload, (response: IEventResponse ) => {
 const insightsActionEventResponse = response as IInsightsActionEventResponse;
 console.log((insightsActionEventResponse.getResponseData() as IInsightsActionData).getInsightsId());
 console.log((insightsActionEventResponse.getResponseData() as IInsightsActionData).getActionId());
 console.log((insightsActionEventResponse.getResponseData() as IInsightsActionData).getActionName());
});
```

Here's a JavaScript example considering recordContext as current browser tabs active record:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
let recordContext = await tabContext.getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsActionEvent');
payload.setId('insightsId1');
insightContext.subscribe(payload, (response) => {
 const insightsActionEventResponse = response;
 console.log(insightsActionEventResponse.getResponseData().getInsightsId());
 console.log(insightsActionEventResponse.getResponseData().getActionId());
 console.log(insightsActionEventResponse.getResponseData().getActionName());
});
```

CTI Agent Assist Features

Show Transcripts During Phone Call

If your CTI supplier provides the live transcription of a phone call in the form of text stream, you can render those transcript messages in the fusion engagement panel.

With this API, you can render the transcript messages for both the agent and the customer. For the transcript messages to be rendered in the engagement panel, you need to enable the transcript feature by setting the transcriptEnabled flag to true using the onToolbarAgentCommand subscription. Once this feature is enabled, you can render customer's messages or agent's messages.

Enable Live Transcripts During a Phone Call

To enable the live transcript feature, you must subscribe to onToolbarAgentCommand event and during getActiveInteractionCommands, create a response object as follows.

Here's an example of the Response object structure for the transcriptEnabled flag:

```
{
 'supportedCommands': [],
 'supportedFeatures': [ // Add feature 'transcriptEnabled' and set isEnabled to true
 {
 'name': 'transcriptEnabled',
 'isEnabled': true
 }
 ]
}
```

Once this feature is enabled, the Notes field in the Fusion Engagement panel is replaced with a transcript component and the messages, "agent joined the call" and "customer joined the call" are shown in the Engagement panel along with a **Call Details** button. Agents can enter notes during a call by clicking the **Call Details** button.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as
 IPhoneContext;
const request = frameworkProvider.requestHelper.createSubscriptionRequest('onToolbarAgentCommand');
phoneContext.subscribe(request, (response) => {
const agentCommandResponse: IMcaOnToolbarAgentCommandEventResponse = response as
 IMcaOnToolbarAgentCommandEventResponse;
return new Promise((resolve, reject) => {
 const agentCommandResponseData: IMcaOnToolbarAgentCommandDataResponse =
 agentCommandResponse.getResponseData();
 const commandObject: IMcaOnToolbarInteractionCommandData =
 agentCommandResponse.getResponseData().getData();
 const command: string = agentCommandResponse.getResponseData().getCommand();
 if (command === 'getActiveInteractionCommands') {
 const outData = {
```

```
 'supportedCommands': [],
 'supportedFeatures': [
 {
 'name': 'transcriptEnabled',
 'isEnabled': true
 }
 ]
 };
 agentCommandResponseData.setOutdata(outData);
 commandObject.result = 'success';
 }
 resolve(commandObject);
 })
})
```

Here's a JavaScript example:

```
const provider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('test', 'v1');

const mcaContext = await provider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');

const request = provider.requestHelper.createSubscriptionRequest('onToolbarAgentCommand');
 phoneContext.subscribe(request, (response) => {
 const agentCommandResponse = response;
 return new Promise((resolve, reject) => {
 const agentCommandResponseData = agentCommandResponse.getResponseData();
 const commandObject = agentCommandResponse.getResponseData().getData();
 const command = agentCommandResponse.getResponseData().getCommand();
 if (command === 'getActiveInteractionCommands') {
 const outData = {
 // For Enabling the features set isEnabled as true
 'supportedFeatures': [
 {
 'name': 'transcriptEnabled',
 'isEnabled': true
 }
 ]
 };
 agentCommandResponseData.setOutdata(outData);
 commandObject.result = 'success';
 }
 resolve(commandObject);
 })
 })
```

Add Live Transcripts of a Customer's Messages During a Phone Call

The FeedLiveTranscript API should be executed from the engagement context, which is available from the startComm event action response. Once the engagement context is available, create the request object for FeeLiveTranscript API and set the required properties (message, message id, state, role). The role should be set as CUSTOMER for adding the customer's transcript messages. Once this API is executed, the customer's transcript message will be rendered in the fusion engagement panel.

Here's a Typescript example:

```
// Step 1: Initialize the framework
```

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');

// Step 2: Get the proper context - engagementContext in this case
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as
 IPhoneContext;
const startCommRequest: IMcaStartCommEventActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('startCommEvent') as IMcaStartCommEventActionRequest;
// TODO set eventId and InData in startComm request
startCommRequest.setAppClassification('ORA_SERVICE');
const startCommOperationResponse: IMcaStartComActionResponse = await phoneContext.publish(startCommRequest)
 as IMcaStartComActionResponse;
const engagementContext: IEngagementContext =
 startCommOperationResponse.getResponseData().getEngagementContext();

// Step 3: Create request object
var requestObject: IMcaFeedLiveTranscriptActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedLiveTranscript') as
 IMcaFeedLiveTranscriptActionRequest;
requestObject.setMessageId('1234');
requestObject.setMessage('Vertical lines on phone display');
requestObject.setRole('END_USER'); // Possible values: AGENT, END_USER
requestObject.setState('CLOSED'); // Possible values: STARTED, INPROGRESS, CLOSED

// Step 4: Execute the API
await engagementContext.publish(requestObject);
```

Here's a JavaScript example:

```
// Step 1: Initialize the framework
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');

// Step 2: Get the proper context - engagementContext in this case
const mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const startCommRequest = frameworkProvider.requestHelper.createPublishRequest('startCommEvent');
// TODO set eventId and InData in startComm request
startCommRequest.setAppClassification('ORA_SERVICE');
const startCommOperationResponse = await phoneContext.publish(startCommRequest);
const engagementContext = startCommOperationResponse.getResponseData().getEngagementContext();

// Step 3: Create request object
var requestObject = frameworkProvider.requestHelper.createPublishRequest('FeedLiveTranscript');
requestObject.setMessageId('1234');
requestObject.setMessage('Vertical lines on phone display');
requestObject.setRole('END_USER'); // Possible values: AGENT, END_USER
requestObject.setState('CLOSED'); // Possible values: STARTED, INPROGRESS, CLOSED

// Step 4: Execute the API
await engagementContext.publish(requestObject);
```

Add System Messages with custom content to the Call Transcripts During a Phone Call

For adding system message for use cases other than join and leave message we can add the properties iconName, message that need to be displayed and type as SYSTEM_MESSAGE.

Note: The get the icon name, go to *Oracle Images*. Not all the icons in the list are supported as system messages.

Here's a Typescript example for adding system messages that use a custom format:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('test', 'v1');
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const engagementContext = getEngagementContext(); //Engagement context received on startcomm response
const feedSystemMessageActionRequest: IMcaFeedSystemMessageActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedSystemMessage') as
 IMcaFeedSystemMessageActionRequest;

feedSystemMessageActionRequest.setEventId(eventId);
feedSystemMessageActionRequest.setAppClassification(FusionHandler.appClassification);

feedSystemMessageActionRequest.setMessageId('1234');
feedSystemMessageActionRequest.setMessage('Test Message');
feedSystemMessageActionRequest.setIconName('oj-ux-ico-target');
feedSystemMessageActionRequest.setType('SYSTEM_MESSAGE');
feedSystemMessageActionRequest.setTime(Date.now());
feedSystemMessageActionRequest.setUserId(userId);
feedSystemMessageActionRequest.setUsername(username);

await engagementContext.publish(feedSystemMessageActionRequest);
```

Here's a JavaScript example for adding system messages that use a custom format:

```
const frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('test', 'v1');
const mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const engagementContext = getEngagementContext(); //Engagement context received on startcomm response
const feedSystemMessageActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedSystemMessage');

feedSystemMessageActionRequest.setEventId(eventId);
feedSystemMessageActionRequest.setAppClassification(FusionHandler.appClassification);

feedSystemMessageActionRequest.setMessageId('1234');
feedSystemMessageActionRequest.setMessage('Test Message');
feedSystemMessageActionRequest.setIconName('oj-ux-ico-target');
feedSystemMessageActionRequest.setType('SYSTEM_MESSAGE');
feedSystemMessageActionRequest.setTime(Date.now());
feedSystemMessageActionRequest.setUserId(userId);
feedSystemMessageActionRequest.setUsername(username);

await engagementContext.publish(feedSystemMessageActionRequest);
```

Add Live Transcripts of Agent's Messages During a Phone Call

You execute the FeedLiveTranscript API from the engagement context, which is available from the startComm event action response.

Once the engagement context is available, create the request object for FeedLiveTranscript API and set the required properties (message, message id, state, role). The role should be set as AGENT for adding the customer's transcript messages. Once this API is executed, the agent's transcript message will be rendered in the fusion engagement panel.

Here's a Typescript example:

```
// Step 1: Initialize the framework
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');

// Step 2: Get the proper context - engagementContext in this case
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as
 IPhoneContext;
const startCommRequest: IMcaStartCommEventActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('startCommEvent') as IMcaStartCommEventActionRequest;
// TODO set eventId and InData in startComm request
startCommRequest.setAppClassification('ORA_SERVICE');
const startCommOperationResponse: IMcaStartComActionResponse = await phoneContext.publish(startCommRequest)
 as IMcaStartComActionResponse;
const engagementContext: IEngagementContext =
 startCommOperationResponse.getResponseData().getEngagementContext();

// Step 3: Create request object
var requestObject: IMcaFeedLiveTranscriptActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedLiveTranscript') as
 IMcaFeedLiveTranscriptActionRequest;
requestObject.setMessageId('5678');
requestObject.setMessage('How can I help you today?');
requestObject.setRole('AGENT'); // Possible values: AGENT, END_USER
requestObject.setState('CLOSED'); // Possible values: STARTED, INPROGRESS, CLOSED

// Step 4: Execute the API
await engagementContext.publish(requestObject);
```

Here's a JavaScript example:

```
// Step 1: Initialize the framework
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');

// Step 2: Get the proper context - engagementContext in this case
const mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const startCommRequest = frameworkProvider.requestHelper.createPublishRequest('startCommEvent');
// TODO set eventId and InData in startComm request
startCommRequest.setAppClassification('ORA_SERVICE');
const startCommOperationResponse = await phoneContext.publish(startCommRequest);
const engagementContext = startCommOperationResponse.getResponseData().getEngagementContext();

// Step 3: Create request object
var requestObject = frameworkProvider.requestHelper.createPublishRequest('FeedLiveTranscript');
requestObject.setMessageId('5678');
requestObject.setMessage('Vertical lines on phone display');
requestObject.setRole('AGENT'); // Possible values: AGENT, END_USER
requestObject.setState('CLOSED'); // Possible values: STARTED, INPROGRESS, CLOSED

// Step 4: Execute the API
await engagementContext.publish(requestObject);
```

Add System Messages to the Call Transcripts During a Phone Call

The FeedSystemMessage API must be executed from the engagement context, which is available from the startComm event action response.

Once the engagement context is available, create the request object for FeedSystemMessage API and set the properties for the type of system message that need to be displayed on the transcript. FeedSystemMessage API support two types of system messages.

Here's some guidelines to bear in mind:

- Predefined system messages for Join or leave messages: You must decide whether a predefined syntax is required to show join or leave message for different use cases.
- System messages for custom implementations: There's no predefined template. You can add any message and choose the icon for the system message.

Add Predefined System Messages to the Call Transcripts During a Phone Call

To add a join or leave message you'll need to set the type of system message to one values shown in the following table:

Also replace the `<name>` property with the actual user name.

Messages

Type	Format
WorkJoin	<name> joined the call.
WorkLeave	<name> left the call.
WorkJoin_Monitor	The supervisor <name> is now observing the call.
WorkLeave_Monitor	The supervisor <name> has stopped observing the call.
WorkJoin_Conferee	The supervisor <name> has joined the call.
WorkLeave_Conferee	The supervisor <name> has left the call.

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('test', 'v1');
const mcaContext: IMultiChannelAdaptorContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const engagementContext = getEngagementContext(); //Engagement context received on startcomm response
const feedSystemMessageActionRequest: IMcaFeedSystemMessageActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedSystemMessage') as
 IMcaFeedSystemMessageActionRequest;

feedSystemMessageActionRequest.setEventId(eventId);
feedSystemMessageActionRequest.setAppClassification(FusionHandler.appClassification);

feedSystemMessageActionRequest.setMessageId('1234);
feedSystemMessageActionRequest.setType('WorkJoin'); // Type of system messages can be any of the value
 mentioned in above table
feedSystemMessageActionRequest.setTime(Date.now());
```

```
feedSystemMessageActionRequest.setUserId(userId);
feedSystemMessageActionRequest.setUsername(username);

await engagementContext.publish(feedSystemMessageActionRequest);
```

Here's a JavaScript example:

```
const frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('test', 'v1');
const mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
const phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
const engagementContext = getEngagementContext(); //Engagement context received on startcomm response
const feedSystemMessageActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('FeedSystemMessage');

feedSystemMessageActionRequest.setEventId(eventId);
feedSystemMessageActionRequest.setAppClassification(FusionHandler.appClassification);

feedSystemMessageActionRequest.setMessageId('1234');
feedSystemMessageActionRequest.setType('WorkJoin'); // Type of system messages can be any of the value
 mentioned in above table
feedSystemMessageActionRequest.setTime(Date.now());
feedSystemMessageActionRequest.setUserId(userId);
feedSystemMessageActionRequest.setUsername(username);

await engagementContext.publish(feedSystemMessageActionRequest);
```

SetToolbarProperties

Publish Action to set toolbar properties

Here's a Typescript example

```
frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID',
 'V1');
let mcaContext: IMultiChannelAdaptorContext;
let phoneContext: IPhoneContext;
let toolBarContext: IToolbarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID',
 'V1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest: ISetToolbarPropertiesActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties') as
 ISetToolbarPropertiesActionRequest;
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((operationResponse: IOperationResponse) => {
```

```
 console.log(operationResponse);
 const setToolbarPropertiesActionResponse: ISetToolbarPropertiesActionResponse = operationResponse as
 ISetToolbarPropertiesActionResponse;
 const setToolbarPropertiesActionResponseData: ISetToolbarPropertiesActionResponseData =
 setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess: boolean = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

Here's a JavaScript example to show Insights, considering recordContext as the current browser tab's active record:

```
let frameworkProvider;
let mcaContextl
let phoneContext;
let toolBarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID',
 'V1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties');
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((setToolbarPropertiesActionResponse) => {
 console.log(setToolbarPropertiesActionResponse);
 const setToolbarPropertiesActionResponseData = setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

SetChannelAvailability

Publish action for set channel availability

Here's a Typescript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

// create request object
const request: ISetChannelAvailabilityRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability') as
 ISetChannelAvailabilityRequest;
request.setAvailability(true);
request.setChannel('PHONE');

// publish request on globalContext
```

```
globalContext.publish(request).then((response: IOperationResponse) => {
 const isSuccess: boolean = (response as ISetChannelAvailabilityResponse).isSuccess();
 if (!isSuccess) {
 console.log('error', (response as ISetChannelAvailabilityResponse).getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'V1');
 const globalContext = await frameworkProvider.getGlobalContext();
 // create request object
 const request = frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability');
 request.setAvailability(true);
 request.setChannel('PHONE');
 // publish request on globalContext
 globalContext.publish(request).then((response) => {
 const isSuccess = response.isSuccess();
 if (!isSuccess) {
 console.log('error', response.getError());
 }
 else {
 console.log('success');
 }
 }).catch(e => console.log(e));
```

4 Modal Window Operations and Events

5 UI Events Framework Reference

IUIEventsFramework

The IUiEventsFramework object is used to initialize the uiEventsFramework by providing an App ID.

All the APIs of the ui-events-framework can be called on top of the object which is available after initializing the ui-events-framework from the client side. To generate that framework provider object, the call to the initialize method supported by IUiEventsFramework object must be made.

Functions

The following code sample shows an example in JavaScript of how to generate the request object for performing an operation initialize.

All the APIs of ui-events-framework can be only called once the framework is initialized from client DOM. The object which we receive by initializing could be used to call those APIs.

The following code sample shows the syntax for initialize API.

```
initialize(applicationName: string, version: string): Promise<IUiEventsFrameworkProvider>;
```

Parameter	Required?	Description
applicationName	Yes	An application name to initialize the framework with.
version	No	A version to initialize the framework with.

The following code sample shows an example in typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

The following code sample shows an example in JavaScript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

IUIEventsFrameworkProvider

The IUiEventsFrameworkProvider is the object on top of which we can call the APIs of the ui-events-framework.

It also provides a requestHelper which can be used to generate the request payload for those APIs. This section explains the methods provided by the IUiEventsFrameworkProvider object. The functions are show the following topics:

getVersion

The getVersion method returns the version with which the framework is initialized.

The following code example shows the syntax for getVersion method:

```
getVersion(): string;
```

The following code example in Typescript shows were the getVersion method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const version : string = frameworkProvider.getVersion();
```

The following code example in Javascript shows where the getVersion method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const version = frameworkProvider.getVersion();
```

getApplicationName

The getApplicationName method returns the application name with which the framework is initialized.

The following code example shows the syntax for getApplicationName function:

```
getApplicationName(): string;
```

The following code example in Typescript shows were the getApplicationName function is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const appName : string = frameworkProvider.getApplicationName(); // appName = 'MyFirstExtensionID'
```

The following code example in Javascript shows were the getApplicationName function is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const appName = frameworkProvider.getApplicationName(); // appName = 'MyFirstExtensionID'
```

getGlobalContext

GlobalContext provides the ability to listen to application-level events and perform application-level operations. For example, listening to context open, context close, and so on.

Use the getGlobalContext function to get the reference to GlobalContext.

The following code example shows the syntax for getGlobalContext method:

```
getGlobalContext(): Promise<IGlobalContext>;
```

The following code example in Typescript shows were the getGlobalContext method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const globalContext: CX_SVC_UI_EVENTS_FRAMEWORK.IGlobalContext = await frameworkProvider.getGlobalContext();
```

The following code example in Javascript shows were the getGlobalContext method is called:

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

getAvailableRecords

This method returns all the valid active objects present in the application.

The following code example shows the syntax:

```
getAvailableRecords(tabId?: string): Promise<IRecordContext[]>;
```

Note: The method will return the active browser's RecordContexts list if browserTabId isn't passed.

The following code example in Typescript shows were the method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const availableRecords: IRecordContext[] = await frameworkProvider.getAvailableRecords();
```

The following code example in Javascript shows were the method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const availableRecords = await frameworkProvider.getAvailableRecords();
```

getCurrentTabId

Returns the current browser tab's tabId.

The following code example shows the syntax:

```
getCurrentTabId(): Promise<string>;
```

The following code example in Typescript shows were the method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const currentTabId: string = await frameworkProvider.getCurrentTabId();
```

The following code example in Javascript shows were the method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const currentTabId = await frameworkProvider.getCurrentTabId();
```

getActiveTabId

Returns the application's active tab.

The following code example shows the syntax:

```
getActiveTab(): Promise<ITabContext>;
```

The following code example in Typescript shows were the method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

```
const activeTab: ITabContext = await frameworkProvider.getActiveTab();
```

The following code example in Javascript shows were the method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const activeTab = await frameworkProvider.getActiveTab();
```

getAvailableTabs

Returns all the available tabs that the application has loaded. It can be MSI tab or a browser tab.

The following code example shows the syntax:

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

The following code example in Typescript shows were the method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const availableTabs: ITabContext[] = await frameworkProvider.getAvailableTabs();
```

The following code example in Javascript shows were the method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const availableTabs = await frameworkProvider.getAvailableTabs();
```

getTabContext

Returns the active tabContext in the application if no optional parameter is passed.

The following code example shows the syntax:

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

Note: The method will return an active TabContext if the browserTabId isn't passed.

browserTabId parameter

Parameter Name	Required	Description
browserTabId	No	TabId for the browser tab which the application has loaded.

The following code example in Typescript shows were the method is called:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
```

The following code example in Javascript shows were the method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
```

```
const tabContext = await frameworkProvider.getTabContext();
```

dispose

This API can dispose or unregister all subscriptions added on the provider object. This method essentially removes all the subscriptions added on globalContext object and recordContext object created from the frameworkProvider object.

```
dispose:() => void;
```

The following code sample shows an example in typescript to dispose all subscriptions from frameworkProvider.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
frameworkProvider.dispose();
```

The following code shows an example in Javascript of how to dispose all subscriptions from frameworkProvider:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 frameworkProvider.dispose();
```

getActiveTabId

Returns the current active tab's context in the application instances.

The following code example shows the syntax:

```
getActiveTab(): Promise<ITabContext>;
```

The following code example in Javascript shows were the method is called:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const activeTabContext:ITabContext = await uiEventsFrameworkInstance.getActiveTab();
```

getCurrentBrowserTabContext

Returns the browser tab context in which the code is executed. The response of this API is always a browser tabContext.

Note: If you call this API from an MCA toolbar window, it will return its parent browser tab's context.

The following code example shows the syntax:

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

The following code example in Javascript shows were the method is called:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const browserTabContext:ITabContext = await getCurrentBrowserTabContext();
```

getModalWindowContext

Part of the UEF provider object used to perform actions on ModalWindow in the Fusion application. You can get the ModalWindowContext by calling getModalWindowContext API provided in UEF provider object.

The following code example shows the syntax:

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

The following code example in Javascript shows were the method is called:

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

getSidePaneContext

Use to get the sidePaneContext object on top of the call to the UEF subscribe and publish APIs to listen to SidePane Open Close events and perform SidePane Expand Collapse, Update (icon update, and section update) actions.

The following code example shows the syntax:

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

The following code example in Javascript shows were the method is called:

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

GetRecordContext

RecordContext allows listening to all events and operations related to an object. For example, listening to field value of a record change either before saving a record event, ot after saving a record event, and so on.

Use the getRecordContext function to get the reference to RecordContext.

The following code example shows the syntax for getRecordContext function:

```
getRecordContext(recordType?: string, recordId?: string): Promise<IRecordContext>;
```

The following table lists the parameters for this function:

Parameter	Required?	Description
recordType	No	The object type. For example, Service Request, Account, and so on.
recordId	No	Unique identifier for the object.

Note: If recordType and recordId values are not supplied,the active Records context is fetched.

The following code sample shows an example in typescript to get the getRecordContext.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const recordContext: CX_SVC_UI_EVENTS_FRAMEWORK.IGlobalContext = await
 frameworkProvider.getRecordContext('SeviceRequest','SeviceRequestNumber');
```

The following code example in Javascript shows were the getRecordContext method is called:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');const recordContext = await
 frameworkProvider.getRecordContext('SeviceRequest','SeviceRequestNumber');
```

getActiveRecord

This method returns the active objects type and unique identifier.

The following code sample shows the syntax for getActiveRecord method:

```
getActiveRecord(): IExtensionContext;
```

The following code sample shows an example in typescript to get the getActiveRecord method.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const activeRecord: CX_SVC_UI_EVENTS_FRAMEWORK.IExtensionContext = frameworkProvider.getActiveRecord();
```

The following code example in Javascript shows were the getActiveRecord method is called:

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

getAvailableRecords

This method returns all the valid active objects present in the application.

The following code sample the syntax for getAvailableRecords method.

```
getAvailableRecords(): IExtensionContext[];
```

The following code sample shows an example in typescript to get the getAvailableRecords method.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const availableRecords: CX_SVC_UI_EVENTS_FRAMEWORK.IExtensionContext[] =
 frameworkProvider.getAvailableRecords();
```

The following code example in Javascript shows getAvailableRecords method:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const availableRecords = frameworkProvider.getAvailableRecords();
```

requestHelper Object

Use this object to generate request objects for events and operations. This request object needs to be furnished to APIs which are used to subscribe to events or perform operations.

Here's the syntax:

```
requestHelper: IRequestHelper;
 IRequestHelper {
 createSubscriptionRequest(eventName: string): IEventRequest;
 createPublishRequest(operationName: string): IOperationRequest;
 }
```

createSubscriptionRequest Function

Use this function to create a requestObject for an event subscription.

browserTabId parameter

Parameter Name	Required	Description
eventName	Yes	Subscribed event name.

The following shows a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');

 globalContext.subscribe(requestObject,(response: IEventResponse) => {
 // custom code
 });
```

The following shows a code example in Javascript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');

 globalContext.subscribe(requestObject,(response) => {
 // custom code
 });
```

createPublishRequest Function

Used to create a requestObject for an event subscription.

createPublishRequest parameter

Parameter Name	Required	Description
operationName	Yes	Operation name that should be subscribed to.

The following shows a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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
});
```

The following shows a code example in Javascript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 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
 });
```

IGlobalContext

IGlobalContext enables you to listen to application-level events and perform application-level operations. For example, listening to context open, context close, and so on.

To get the reference to GlobalContext, you can use the getGlobalContext function. This function provides a reference to the application globalContext.

The following code sample shows the syntax for getGlobalContext method:

```
getGlobalContext(): Promise<IGlobalContext>;
```

The following is a code sample Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const globalContext = await frameworkProvider.getGlobalContext();
```

Functions

Here's a list of the supported functions:

- subscribe
- subscribeOnce
- publish
- dispose
- getSupportedEvents
- getSupportedActions

subscribe

Use this API to subscribe to an event from the Fusion application. Using this API, the external application can listen to the application level events from the the Fusion application. Once an event is subscribed using this API, an external application will be notified until the subscription is disposed.

The following code sample shows the syntax for subscribe:

```
subscribe: (payload: IEventSubscriptionPayload, callbackFunction: (response:IEventResponsePayload) => void)
=> ISubscriptionContext;
```

Here are the parameters:

Parameter	Required?	Description
payload	Yes	Request object for the event being subscribed to.
callbackFunction	Yes	A callback function where we get the event response as its arguments.

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 const subscriptionContext: ISubscriptionContext = globalContext.subscribe(payload, (response:
 IEventResponse) => {
 // custom code
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 const subscriptionContext = globalContext.subscribe(payload, (response) => {
 // custom code
 });
```

subscribeOnce

Use this API to subscribe to a the Fusion application event just once. Using this API, external applications can listen to the application level events from the the Fusion application. But this subscription will be disposed of automatically after the first notification.

The following code sample shows the syntax for subscribeOnce:

```
subscribeOnce: (payload: IEventSubscriptionPayload, callbackFunction: (response: IEventResponsePayload) =>
 void => ISubscriptionContext;
```

Here are the parameters:

Parameter	Required?	Description
payload	Yes	Request object for the event being subscribed to.
callbackFunction	Yes	A callback function where we get the event response as its arguments.

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 const subscriptionContext: ISubscriptionContext = globalContext.subscribeOnce(payload, (response:
 IEventResponse) => {
 // custom code
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 const subscriptionContext = globalContext.subscribeOnce(payload, (response) => {
 // custom code
 });
```

publish

This API can inform the Fusion application to operate. The API returns a promise to which you can add a then-catch block. The then block would return the operation's status and any data returned after the process is completed.

The following code sample shows the syntax for subscribeOnce:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

```
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusCustomEventOperation');
 globalContext.publish(payload).then((response: IOperationResponse)=>{
 // custom code
 }).catch((error)=>{
 // error
 })
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const payload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusCustomEventOperation');
globalContext.publish(payload).then((response) => {
 // custom code
}).catch((error) => {
 // error
})
```

dispose

This API can dispose or unregister any subscriptions that's added for an event. By disposing the subscription, the API will not trigger any further notifications to the client side receiver.

```
dispose:() => void;
```

The following code sample shows an example in typescript for dispose API.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: CX_SVC_UI_EVENTS_FRAMEWORK.IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
const subscriptionContext: CX_SVC_UI_EVENTS_FRAMEWORK.ISubscriptionContext =
 globalContext.subscribe(requestObject, (response: CX_SVC_UI_EVENTS_FRAMEWORK.IEventResponse) => {
 // custom code
 });
globalContext.dispose();
```

The following code sample shows an example in Javascript for dispose API.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
const subscriptionContext = globalContext.subscribe(requestObject, (response:
 CX_SVC_UI_EVENTS_FRAMEWORK.IEventResponse) => {
// custom code
});
globalContext.dispose();
```

getSupportedEvents

This method returns all the supported events on top of globalContext.

The following code sample shows the syntax for getSupportedEvents:

```
getSuportedEvents(): string[];
```

The following code sample shows example for getSupportedEvents in Javascript.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: CX_SVC_UI_EVENTS_FRAMEWORK.IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const supportedEvents= globalContext.getSupportedEvents();
```

getSupportedActions

This method returns all the supported actions on top of globalContext.

The following code sample shows the syntax for getSupportedActions:

```
getSupportedActions(): string[];
```

The following code sample shows example for getSupportedActions in Javascript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const supportedActions: string[] = globalContext.getSupportedActions();
```

getGlobalContextId

This method returns all unique context id of the GlobalContext object..

The following code sample shows the syntax for getGlobalContextId:

```
getGlobalContextId(): string;
```

The following code sample shows example for getSupportedActions in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const globalContextId = globalContext.getGlobalContextId();
```

UI Events Framework actions on GlobalContext for multiple browser tabs

UI Events Framework supports four different contexts such as GlobalContext, TabContext and RecordContexts and PhoneContext. If you listen to an event that's supported on GlobalContext, you can get its notification whenever this event is originated from any of the browser tabs which are loaded with the application.

However, for actions, an action performed on top of globalContext,would be executed only on the current browser tab. So, there's different behavior with globalContext for events and for actions.

There are also some use cases where you could perform an action on every browser tab. This might not return the results you expect. Using CustomEvent, PopUp, Modal and GetAgentInfo are the actions supported on GlobalContext can perform in multiple browser tabs in a single action publish.

Publishing an action from globalContext works in the following way.

1. If there are multiple browser tabs opened, then the operation request is sent to all the browser tabs if user sets the setPropagateToTabs method to True
2. The client who requested this action will get positive feedback only when all the browser tabs successfully mark this request as completed.

3. The client then receives an array of IOperationResponse in the publish request's then-block, instead of getting a single response as before.
4. Each entry in the OperationResponse array will contain the context of each browser tab in which it was executed and the corresponding response data.
5. The catch block is executed when any one of the browser tab rejects this action request.

> **Note:** The response structure will be same as in previous releases, for actions in TabContext and RecordContext. Only GlobalContext level action response will be changed to array of IOperationResponse.

ITabContext

Enables you to listen to application-level events and perform application-level operations from a specific browser tab or MSI tab. For example, listening to context open, context close, and so on.

To get the reference to TabContext, you can use the getTabContext function. This function provides a reference to the active tab in which the application is loaded. You can pass the browserTabId as an optional parameter to get a particular browser tab's context.

Here's the syntax:

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

Parameter

Parameter Name	Required?	Description
browserTabId	No	A particular browser tab's ID. **Note:** You can get each browser's tabId by calling the getCurrentTabId API on the frameworkProvider from each browser tab.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider : IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
const tabContext = await frameworkProvider.getTabContext();
```

getType Function

Returns the type of a tabContext object. The type can be Browser Tab, MSI tab or MSI Sub Tab.

The following block shows the syntax for getType:

```
getType(): string;
```

The following code sample shows an example in typescript of getType.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const type: string = tabContext.getType();
```

The following code sample shows an example in Javascript of getType.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const type = tabContext.getType();
```

subscribe

Use this API to subscribe to an event from the Fusion application. Using this API, the external application can listen to the application level events from the the Fusion application. Once an event is subscribed using this API, an external application will be notified until the subscription is disposed.

The following code sample shows the syntax for subscribe:

```
subscribe: (payload: IEventSubscriptionPayload, callbackFunction: (response:IEventResponsePayload) => void)
=> ISubscriptionContext;
```

Here are the parameters:

Parameter	Required?	Description
payload	Yes	Request object for the event being subscribed to.
callbackFunction	Yes	A callback function where we get the event response as its arguments.

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 const subscriptionContext: ISubscriptionContext = tabContext.subscribe(payload, (response: IEventResponse)
 => {
 // custom code
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
```

```
const tabContext = await frameworkProvider.getTabContext();
const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
const subscriptionContext = tabContext.subscribe(payload, (response) => {
// custom code
});
```

subscribeOnce

Use this API to subscribe to the Fusion event just once. Using this API, external applications can listen to the application level events from the Fusion application. But this subscription will be disposed of automatically after the first notification.

The following code sample shows the syntax:

```
subscribeOnce: (payload: IEventSubscriptionPayload, callbackFunction: (response:
 IEventResponsePayload) => void) => ISubscriptionContext;
```

Here are the parameters:

Parameter	Required?	Description
payload	Yes	Request object for the event being subscribed to.
callbackFunction	Yes	A callback function where we get the event response as its arguments.

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 const subscriptionContext = tabContext.subscribeOnce(payload, (response: IEventResponse) => {
 // custom code
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 const subscriptionContext = tabContext.subscribeOnce(payload, (response) => {
 // custom code
 });
```

publish

This API can inform the Fusion application to operate. The API returns a promise to which you can add a then-catch block. The then block would return the operation's status and any data returned after the process is completed.

The following code sample shows the syntax for subscribeOnce:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabs: ITabContext[] = await frameworkProvider.getAvailableTabs();
 const tabContext: ITabContext = tabs[0];
 const payload: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const response: IFocusTabResponseData = message as IFocusTabResponseData;
 const currentTab: ITabContext = response.getResponseData().getCurrentTab();
 const previousTab: ITabContext = response.getResponseData().getPreviousTab()
 }).catch((error: IErrorData) => {
 // error
 });
```

The following is a code sample in JavaScript:

```
const tabs = await frameworkProvider.getAvailableTabs();
 const tabContext = tabs[0];
 const payload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((response) => {
 const currentTab = response.getResponseData().getCurrentTab();
 const previousTab = response.getResponseData().getPreviousTab()
 }).catch((error) => {
 // error
 });
```

dispose

This API can dispose or unregister any subscriptions that's added for an event. By disposing the subscription, the API will not trigger any further notifications to the client side receiver.

```
dispose:() => void;
```

The following code sample shows an example in typescript for dispose API.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 const subscriptionContext: ISubscriptionContext = tabContext.subscribe(payload, (response: IEventResponse)
 => {
 // custom code
 });
 tabContext.dispose();
```

The following code sample shows an example in Javascript for dispose API.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 const subscriptionContext = tabContext.subscribe(payload, (response) => {
 // custom code
 });
 tabContext.dispose();
```

getSupportedEvents

This method returns all the supported events on top of globalContext.

The following code sample shows the syntax for getSupportedEvents:

```
getSuportedEvents(): string[];
```

The following code sample shows example for getSupportedEvents in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const supportedEvents: string[] = tabContext.getSupportedEvents();
```

The following code sample shows example for getSupportedEvents in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const supportedEvents = tabContext.getSupportedEvents();
```

getSupportedActions

This method returns all the supported actions on top of globalContext.

The following code sample shows the syntax for getSupportedActions:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const supportedActions: string[] = tabContext.getSupportedActions();
```

The following code sample shows example for getSupportedActions in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const supportedActions = tabContext.getSupportedActions();
```

getAvailableTabs

This method returns all the available tabs on top of a specific tab or its tabContext.

The following code sample shows the syntax:

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

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
```

```
const availableTabs: ITabContext[] = await tabContext.getAvailableTabs();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const availableTabs = await tabContext.getAvailableTabs();
```

getActiveRecord

This method returns all the active records on a specific tab.

The following code sample shows the syntax:

```
getActiveRecord(): Promise<IRecordContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const activeRecord: IRecordContext = await tabContext.getActiveRecord();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const activeRecord = await tabContext.getActiveRecord();
```

getAvailableRecord

This method returns all the available records on a specific tab.

The following code sample shows the syntax:

```
getAvailableRecords(): Promise<IRecordContext[]>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const availableRecords: IRecordContext[] = await tabContext.getAvailableRecords();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const dependentTabs = await tabContext.getDependentTabs();
```

getTabInfo

This method returns the tab ID, MSI tab ID and MSI tab ID associated with the TabContext.

The following code sample shows the syntax:

```
getTabInfo(): ITabInfo;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const tabInfo: ITabInfo = tabContext.getTabInfo();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext = await frameworkProvider.getTabContext();
const tabInfo = await tabContext.getTabInfo();
```

getEngagementInfo

This method returns the engagements associated with the TabContext.

The following code sample shows the syntax:

```
getEngagementInfo(): Promise<IEngagementInfo>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const engagementInfo: IEngagementInfo = await tabContext.getEngagementInfo();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext = await frameworkProvider.getTabContext();
const engagementInfo = await tabContext.getEngagementInfo();
```

IRecordContext

IRecordContext extends the functionalities of IContext and provides RecordType and RecordId.

Here's the interface:

```
export interface IRecordContext extends IContext {
 getRecordType(): string;
 getRecordId(): string;
```

```
}
```

Functions

subscribe

Use this API to subscribe to an event from the Service Center. Using this API, the external application is able to listen to the object level events from the Service Center. Once an event is subscribed using this API, an external application will be notified until the subscription is disposed.

The following sample shows the syntax for subscribe:

```
subscribe: (requestObject: IEventRequest, callbackFunction: (response: IEventResponse) => void) =>
 ISubscriptionContext;
```

The following is a code sample in Typescript.

```
const requestObject: IFieldValueChangeEventRequest =
 this.frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent') as
 IFieldValueChangeEventRequest;
 requestObject.setFields(['objectType.fieldnameOne', 'objectType.fieldnameTwo',
 'objectType.fieldnameThree']);
 let subscription: ISubscriptionContext = await recordContext.subscribe(requestObject, (eventResponse:
 IEventResponse) => {
 // eventResponse
 });
```

The following is a code sample in Javascript.

```
const requestObject =
 this.frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent');
 requestObject.setFields(['objectType.fieldnameOne', 'objectType.fieldnameTwo',
 'objectType.fieldnameThree']);
 let subscription = await recordContext.subscribe(requestObject, (eventResponse) => {
 // eventResponse
 });
```

subscribeOnce

Subscribe once to a record level event from the Fusion application. For example, a Field Value Change event. The subscription is disposed once the event is fired and the call back is executed

The following code sample shows the syntax:

```
subscribeOnce: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse)
 => void) => ISubscriptionContext;
```

The following is a code sample in Typescript:

```
const requestObject: IFieldValueChangeEventRequest =
 this.frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent') as
 IFieldValueChangeEventRequest;
 requestObject.setFields(['objectType.fieldnameOne', 'objectType.fieldnameTwo',
 'objectType.fieldnameThree']);
 let subscription: ISubscriptionContext = await recordContext.subscribeOnce(requestObject, (eventResponse:
 IEventResponse) => {
 // eventResponse
 });
```

The following is a code sample in JavaScript:

```
const requestObject =
 this.frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent');
 requestObject.setFields(['objectType.fieldnameOne', 'objectType.fieldnameTwo',
 'objectType.fieldnameThree']);
 let subscription = await recordContext.subscribeOnce(requestObject, (eventResponse) => {
 // eventResponse
 });
```

publish

Publish an operation at the record Context. For example, set a value of a particular field.

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
const requestObject: ISetFieldValueOperationRequest =
 this.frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest;
 requestObject.field().setValue('objectType.fieldName', 'a value that matches type of field name');
 recordContext.publish(requestObject).then((response: IOperationResponse) => {
 console.log((response as ISetFieldValueResponse).getResponseData().getMessage());
 }).catch((error) => console.log(error));
```

The following is a code sample in JavaScript:

```
const requestObject =
 this.frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 requestObject.field().setValue('objectType.fieldName', 'a value that matches type of field name');
 recordContext.publish(requestObject).then((response) => {
 console.log(response.getResponseData().getMessage());
 }).catch((error) => console.log(error));
```

dispose

Subscribe to a record level event from the Fusion application. For example, a Field Value Change event.

```
dispose:() => void;
```

The following code sample shows an example in Typescript and JavaScript:

```
recordContext.dispose();
```

getSupportedEvents

Get the events supported in a record context as an array of strings

The following code sample shows the syntax:

```
getSupportedEvents(): string[];
```

The following code sample shows example in Typescript.

```
const supportedEvents: string[] = recordContext.getSupportedEvents();
```

The following code sample shows example in Javascript.

```
const supportedEvents = recordContext.getSupportedEvents();
```

getSupportedActions

Get the actions supported in a record context as an array of strings.

The following code sample shows the syntax:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
const supportedActions: string[] = recordContext.getSupportedActions();
```

The following code sample shows example for getSupportedActions in Javascript.

```
const supportedActions = recordContext.getSupportedActions();
```

getRecordType

Returns the record type of the loaded object in the record context.

The following code sample shows the syntax:

```
getRecordType(): string;
```

The following code sample shows an example in Typescript.

```
const recordType: string = recordContext.getRecordType();
```

The following code sample shows an example in Javascript.

```
const recordType = recordContext.getRecordType();
```

getRecordId

Returns the record ID. For a record context of a record not created yet, the ID will be a negative number.

The following code sample shows the syntax:

```
getRecordId(): string;
```

The following code sample shows an example in typescript to getRecordId:

```
const recordId: string = recordContext.getRecordId();
```

The following code sample shows an example in JavaScript to getRecordId:

```
const recordId = recordContext.getRecordId();
```

ISidePaneContext

This is the object which encapsulates all the UI Events Framework-provided SidePane events and actions. This object only acts on UIEF advanced side panes. The legacy side pane doesn't support the events and actions supported by this object.

To get the reference to SidePaneContext, you can use the getSidePaneContext function. This function provides a reference to the sidePaneContext of a particular sidePaneId.

Note: SidePaneId denotes the ID with which user enabled the advanced side panes (multiple side panes) in UEF.

Here's the syntax:

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

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const sidePaneContext: ISidePaneContext = await frameworkProvider.getSidePaneContext('sidepane-id');
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const sidePaneContext = await frameworkProvider.getSidePaneContext();
```

Functions

subscribe

Use this function to listen to a SidePaneContext supported event.

The following sample shows the syntax:

```
subscribe: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse) =>
 void) => ISubscriptionContext;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 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();
 });
 }
```

The following is a code sample in Javascript.

```
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, (res) => {
 const response = res as ISidePaneOpenEventResponse;
 const responseData = response.getResponseData();
 console.log(responseData.getActiveSectionId());
 });
```

```
}
```

subscribeOnce

Use this function to listen once to a SidePaneContext supported event.

The following code sample shows the syntax:

```
subscribeOnce: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse)
 => void) => ISubscriptionContext;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 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.subscribeOnce(payload, (res: IEventResponse) => {
 const response = res as ISidePaneOpenEventResponse;
 const responseData: ISidePaneData = response.getResponseData(); responseData.getActiveSectionId();
 });
 }
```

The following is a code sample in JavaScript:

```
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.subscribeOnce(payload, (res) => {
 const response = res as ISidePaneOpenEventResponse;
 const responseData = response.getResponseData();
 console.log(responseData.getActiveSectionId());
 });
 }
```

publish

Use this function to publish a SidePane supported operation on the SidePaneContext object.

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 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'); payload.setIcon('newIcon'); payload.setVisibility(true);
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response);
 }).catch(() => { })
```

```
}
```

The following is a code sample in JavaScript:

```
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'); payload.setIcon('newIcon'); payload.setVisibility(true);
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response);
 }).catch(() => { })
 }
```

dispose

Use this function to remove all events subscribed on SidePaneContext object.

```
dispose:() => void;
```

The following code sample shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const disposeSidePaneEvents = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 sidePaneContext.dispose();
 }
```

The following code sample shows an example in JavaScript:

```
const disposeSidePaneEvents = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 sidePaneContext.dispose();
 }
```

getSupportedEvents

Use this function to get all the supported events on the SidePaneContext object.

The following code sample shows the syntax:

```
getSupportedEvents(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getSupportedEvents = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const supportedEvents: string[] = sidePaneContext.getSupportedEvents();
 }
```

The following code sample shows example in Javascript.

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

```
const supportedEvents = sidePaneContext.getSupportedEvents();
}
```

getSupportedActions

Use this function to get all the supported actions on the SidePaneContext object.

The following code sample shows the syntax:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getSupportedActions = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const supportedActions: string[] = sidePaneContext.getSupportedActions();
 }
```

The following code sample shows example for getSupportedActions in Javascript.

```
const getSupportedActions = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const supportedActions = sidePaneContext.getSupportedActions();
 }
```

getSidePaneContextId

Use this function to get the sidePane contextId of the SidePaneContext object. .

The following code sample shows the syntax:

```
getSidePaneContextId(): string;
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getContextId = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const contextId: string = sidePaneContext.getSidePaneContextId();
 }
```

The following code sample shows example for getSupportedActions in Javascript.

```
const getContextId = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const contextId = sidePaneContext.getSidePaneContextId();
 }
```

IModalWindowContext

This is the object which encapsulates all the UI Events Framework and provides modal and pop up window actions.

To get the reference to ModalWindowContext, you can use the getModalWindowContext function. This function provides a reference to the modalWindowContext.

Here's the syntax:

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

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const modalWindowContext: IModalWindowContext = await frameworkProvider.getModalWindowContext();
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID', 'v1');
 const modalWindowContext = await frameworkProvider.getModalWindowContext();
```

Functions

publish

Use this function to publish a publish a show modal operation with ModalWindowContext object.

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://test.oracle.com:8080/test.html');
 requestObject.setId('modal_103');
 const response: IModalWindowOperationResponse = await modalWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
 }
```

The following is a code sample in JavaScript:

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

```
const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal')
requestObject.setURL('https://test.oracle.com:8080/test.html');
requestObject.setId('modal_103');
const response = await modalWindowContext.publish(requestObject);
const id = response.getResponseData().getId();
}
```

getSupportedActions

Use this function to get get all the supported actions on ModalWindowContext object.

The following code sample shows the syntax:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getSupportedActions = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const supportedActions: string[] = sidePaneContext.getSupportedActions();
 }
```

The following code sample shows example for getSupportedActions in Javascript.

```
const getSupportedActions = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const supportedActions = modalWindowContext.getSupportedActions();
 }
```

INotificationContext

INotificationContext enables you to do actions and events on Notifications.For example, show Notification, Close Notification, Listen to close notification, Listen to actions in notification.

To get the reference to NotificationContext use the getNotificationContext function.ds

Here's the syntax:

```
getNotificationContext(notificationId: string): Promise<INotificationContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId');
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
```

```
const notificationContext = await frameworkProvider.getNotificationContext('notificationId');
```

Functions

subscribe

Use this function to listen to a SidePaneContext supported event.

The following sample shows the syntax:

```
subscribe: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse) =>
 void) => ISubscriptionContext;
```

The following is a code sample in Typescript.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent') as
 IEventRequest;
 notificationContext.subscribe(requestObject, (response) => {
 const resp: INotificationCloseActionEventResponse = response as INotificationCloseActionEventResponse;
 console.log(resp.getResponseData().getNotificationId())
 });
```

The following is a code sample in Javascript.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent');
 notificationContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId())
 });
```

subscribeOnce

Use this function to listen once to a close event.

The following code sample shows the syntax:

```
subscribeOnce: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse)
 => void) => ISubscriptionContext;
```

The following is a code sample in Typescript:

```
//// <reference path="uiEventsFramework.d.ts"/> const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const requestObject:IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent') as
 IEventRequest;
 notificationContext.subscribeOnce(requestObject, (response) => {
 const resp: INotificationCloseActionEventResponse = response as INotificationCloseActionEventResponse;
 console.log(resp.getResponseData().getNotificationId())
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnNotificationCloseActionEvent');
 notificationContext.subscribeOnce(requestObject, (response) => {
 console.log(response.getResponseData().getNotificationId())
 });
```

publish

Use this function to publish a supported operation on the NotificationContext object.

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/> const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const payload: IShowNotificationRequest =
 frameworkProvider.requestHelper.createPublishRequest('ShowNotification') as IShowNotificationRequest)
 payload.setTitle(fieldName);
 payload.setSummary('Error message summary');
 notificationContext.publish(requestObject).then((message) => {
 const response: INotificationOperationResponse = message as INotificationOperationResponse;
 //custom code
 });
```

The following is a code sample in JavaScript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 let notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 const payload = frameworkProvider.requestHelper.createPublishRequest('ShowNotification');
 payload.setTitle(fieldName);
 payload.setSummary('error message summary ');
 notificationContext.publish(payload).then((response) => {
 // custom code
 }).catch((err) => {
 console.log(err);
 });
```

dispose

Use this function to remove all events subscribed on NotificationContext object.

```
dispose:() => void;
```

The following code sample shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const disposeSidePaneEvents = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 notificationContext.dispose();
```

```
}
```

The following code sample shows an example in JavaScript:

```
const disposeSidePaneEvents = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const notificationContext = await frameworkProvider.getNotificationContext('notificationId007');
 notificationContext.dispose();
 }
```

getSupportedEvents

Use this function to get all the supported events on the NotificationContext object.

The following code sample shows the syntax:

```
getSupportedEvents(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getSupportedEvents = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const supportedEvents: string[] = sidePaneContext.getSupportedEvents();
 }
```

The following code sample shows example in Javascript.

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

getSupportedActions

Use this function to get all the supported actions on the NotificationContext object.

The following code sample shows the syntax:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const getSupportedActions = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const notificationContext: INotificationContext = await
 frameworkProvider.getNotificationContext('notificationId007');
 const supportedActions: string[] = notificationContext.getSupportedActions();
 }
```

The following code sample shows example for getSupportedActions in Javascript.

```
const getSupportedActions = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const notificationContext = await uiEventsFrameworkInstance.getNotificationContext('notificationId007');
 const
```

IMultiChannelAdaptorContext

The IMultiChannelAdaptorContext object encapsulates the events and actions on different channels when a telephonic or chat interaction occurs.

This object is added on top of UIEF provider object to access a CommunicationChannelContext. For example, PhoneContext. You can get the MultiChannelAdaptorContext by calling the getMultiChannelAdaptorContext API provided in UEF provider object.

Here's the syntax:

```
getMultiChannelAdaptorContext(): Promise<IMultiChannelAdaptorContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

Functions

getActiveChannels

Use this function to get the supported Channels. For example, PHONE or CHAT.

The following sample shows the syntax:

```
getActiveChannels(): Promise<string[]>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const activeChannels: string[] = await multiChannelAdaptorContext.getActiveChannels();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const activeChannels = await multiChannelAdaptorContext.getActiveChannels();
```

getCommunicationChannelContext

Use this function to get the context of the communication channel supported over multiChannelAdaptorContext object. For example, PhoneContext.

The following sample shows the syntax:

```
getCommunicationChannelContext(channelType: string):
 Promise<ICommunicationChannelContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
```

IPhoneContext

PhoneContext is the object that encapsulates the UIEF-provided telephonic events and act icons. This object can be derived from MultiChann elAdaptorContext object by calling getCommunicationChannelContext API by passing channelType as PHONE.

Here's the syntax:

```
getCommunicationChannelContext(channelType: McaChannels): Promise<ICommunicationChannelContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
```

Functions

subscribe

Use this function toto listen to a PhoneContext supported Event. For example, onToolbarInteractionCommand, onDataUpdated, onOutgoingEvent.

The following sample shows the syntax:

```
subscribe: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse) =>
 void) => ISubscriptionContext;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated') as IMcaEventRequest;
 request.setAppClassification('_appClassfication');
 phoneContext.subscribe(request, (response: IEventResponse) => {
 const dataUpdateResponse = response as IMcaOnDataUpdatedEventResponse;
 const data: IMcaOnDataUpdatedData = dataUpdateResponse.getResponseData();
 });
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated') as
 IMcaEventRequest;
 request.setAppClassification('_appClassfication');
 phoneContext.subscribe(request, (response) => {
 const data = response.getResponseData();
 });
```

subscribeOnce

Use this function to listen only once to a PhoneContext supported Event. For example,. onToolbarInteractionCommand, onDataUpdated, onOutgoingEvent.

The following code sample shows the syntax:

```
subscribeOnce: (requestObject: IEventRequest, callbackFunction: (response:IEventResponse)
 => void) => ISubscriptionContext;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
```

```
const request: IMcaEventRequest =
uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated') as IMcaEventRequest;
request.setAppClassification('_appClassfication');
phoneContext.subscribeOnce(request, (response: IEventResponse) => {
const dataUpdateResponse = response as IMcaOnDataUpdatedEventResponse;
const data: IMcaOnDataUpdatedData = dataUpdateResponse.getResponseData();
});
```

The following is a code sample in JavaScript:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onDataUpdated') as
IMcaEventRequest;
request.setAppClassification('_appClassfication');
phoneContext.subscribeOnce(request, (response) => {
const data = response.getResponseData();
});
```

publish

Use this function to publish a PhoneContext-supported operation on PhoneContext object..

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

The following is a code sample in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
const request: IMcaNewCommEventActionRequest =
uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent') as
IMcaNewCommEventActionRequest;
request.setAppClassification('self.appClassfication');
request.setLookupObject('lookup');
request.setInputData(_inboundData);// _inboundData of type - IMcaNewCommInData
const inData: IMcaInDataRequest = request.getInData();
inData.setCallStatus('INCOMING');
phoneContext.publish(request).then((response: IOperationResponse) => {
const newCommResponse: IMcaNewComActionResponse = response as IMcaNewComActionResponse;
}).catch(()=>{
});
```

The following is a code sample in JavaScript:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent');
request.setAppClassification('self.appClassfication');
request.setLookupObject('lookup');
request.setInputData(_inboundData);// _inboundData of type - IMcaNewCommInData
const inData = request.getInData();
inData.setCallStatus('INCOMING');
phoneContext.publish(request).then((response) => {
```

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

dispose

Use this function to remove all events subscribed subscribed on the PhoneContext object.

```
dispose:() => void;
```

The following code sample shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 phoneContext.dispose();
```

The following code sample shows an example in JavaScript:

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 phoneContext.dispose();
```

getSupportedEvents

Use this function to get all the supported events on the PhoneContext object.

The following code sample shows the syntax:

```
getSupportedEvents(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const supportedEvents:string[] = phoneContext.getSupportedEvents();
```

The following code sample shows example in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const supportedEvents = phoneContext.getSupportedEvents();
```

getSupportedActions

Use this function to get all the supported actions on the PhoneContext object.

The following code sample shows the syntax:

```
getSupportedActions(): string[];
```

The following code sample shows example in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const supportedActions: string[] = phoneContext.getSupportedActions();
```

The following code sample shows example for getSupportedActions in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const supportedActions = phoneContext.getSupportedActions();
```

getChannelType

Use this function to get the channel type in PhoneContext API requests. For example, ORA_SVC_PHONE.

The following sample shows the syntax:

```
getChannelType(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const channelType: string = phoneContext.getChannelType();
```

The following is a code sample in Javascript.

```
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const channelType = phoneContext.getChannelType();
```

getChannel

Use this function to get the type in context object. The value will be PHONE.

The following sample shows the syntax:

```
getChannel(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
```

```
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
const channel: string = phoneContext.getChannel(); // 'PHONE'
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const channel = phoneContext.getChannel(); // 'PHONE'
```

getFrameOrigin

Use this function to get the frame origin in PhoneContext object requests.

The following sample shows the syntax:

```
getFrameOrigin(): string;
```

The following is a code sample in Typescript.

```
//// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const frameOrigin: string = phoneContext.getFrameOrigin();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const frameOrigin = phoneContext.getFrameOrigin();
```

getToolbarName

Use this function to get the toolbar name in PhoneContext object request

The following sample shows the syntax:

```
getToolbarName(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const toolbarName: string = phoneContext.getToolbarName();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const toolbarName = phoneContext.getToolbarName();
```

getVersion

Use this function to get the version in PhoneContext object reques.

The following sample shows the syntax:

```
getVersion(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const version: string = phoneContext.getVersion();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const version = phoneContext.getVersion();
```

getEventSource

Use this function to get the event source in PhoneContext object requests.

The following sample shows the syntax:

```
getEventSource(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const eventSource: string = phoneContext.getEventSource();
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
```

```
const eventSource = phoneContext.getEventSource();
```

IEngagementContext

EngagementContext is the object that encapsulates the tabContext in which the new call connection has been established because of a startCommEvent publish request. As a result, you can get control over the objects being opened by that call connection.

Here's the syntax:

```
getEngagementContext(): IEngagementContext;
```

The following is a code sample in Typescript.

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;
const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
 //You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('appClassfication');
request.setEventId('1'); // Set the event ID here
const inData: IMcaStartCommInDataRequest = request.getInData();
phoneContext.publish(request).then((res: IOperationResponse) => {
 const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
 const startCommData: IMcaStartComActionData = response.getResponseData();
 const engagementContext: IEngagementContext = startCommData.getEngagementContext();
})
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
 //You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData = request.getInData();
 phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
 })
```

Functions

getEngagementId

Use this function to get the unique identifier of the engagement object opened because of a new call connection

The following sample shows the syntax:

```
getEngagementId(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
 //You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData: IMcaStartCommInDataRequest = request.getInData();
 phoneContext.publish(request).then((res: IOperationResponse) => {
 const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
 const startCommData: IMcaStartComActionData = response.getResponseData();
 const engagementContext: IEngagementContext = startCommData.getEngagementContext();
 const engagementId: string = engagementContext.getEngagementId();
 }).catch(() => {
 })
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
 // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData = request.getInData();
 phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
 const engagementId = engagementContext.getEngagementId();
 })
```

getTabContext

Use this function to get the tabContext on which the engagement object or new call connection is started. This will be of type ITabContext and you can get the active and available records on this tabContext and perform actions on it.

The following sample shows the syntax:

```
getTabContext(): Promise<ITabContext>;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
 request.setAppClassification('appClassfication');// You should set correct appClassification here. eg:
 ORA_SERVICE for Service App
 request.setEventId('1'); // Set the event ID here
 const inData: IMcaStartCommInDataRequest = request.getInData();
 phoneContext.publish(request).then(async (res: IOperationResponse) => {
 const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
 const startCommData: IMcaStartComActionData = response.getResponseData();
 const engagementContext: IEngagementContext = startCommData.getEngagementContext();
 const tabContext: ITabContext = await engagementContext.getTabContext();
 }).catch(() => {
 })
```

IEngagementContext IFieldValueChangeEventRequest

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
 request.setAppClassification('appClassfication');// You should set correct appClassification here. eg:
 ORA_SERVICE for Service App
 request.setEventId('1'); // Set the event ID here
 const inData = request.getInData();
 phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
 const engagementId = engagementContext.getEngagementId();
 const tabContext = await engagementContext.getTabContext();
 })
```

getEngagementData

Use this function to fetch the data in the engagement object.

The following sample shows the syntax:

```
getEngagementData(): IMcaEngagementData;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
```

```
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
 // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData: IMcaStartCommInDataRequest = request.getInData();
 phoneContext.publish(request).then(async (res: IOperationResponse) => {
 const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
 const startCommData: IMcaStartComActionData = response.getResponseData();
 const engagementContext: IEngagementContext = startCommData.getEngagementContext();
 const engagementData: IMcaEngagementData = await engagementContext.getEngagementData();
 }).catch(() => {
 })
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
 // You should set correct appClassification here. eg: ORA_SERVICE for Service App
// Refer https://docs.oracle.com/en/cloud/saas/fusion-service/fairs/application-classification-
code.html#s20059918 for the list of supported app classifications
request.setAppClassification('appClassfication');
request.setEventId('1'); // Set the event ID here
const inData = request.getInData();
phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
 const engagementId = engagementContext.getEngagementId();
 const engagementData = await engagementContext.getEngagementData();
})
```

getChannel

Use this function to get the type in EngagementContext object. Value will be PHONE.

The following sample shows the syntax:

```
getChannel(): string;
```

The following is a code sample in Typescript.

```
/// <reference path="uiEventsFramework.d.ts"/> const uiEventsFrameworkInstance: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
 // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer to Application Classification Code for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData: IMcaStartCommInDataRequest = request.getInData();
 phoneContext.publish(request).then(async (res: IOperationResponse) => {
 const response: IMcaStartComActionResponse = res as IMcaStartComActionResponse;
```

```
const startCommData: IMcaStartComActionData = response.getResponseData();
const engagementContext: IEngagementContext = startCommData.getEngagementContext();
const channel: string = await engagementContext.getChannel(); // 'PHONE'
}).catch(() => {
})
```

The following is a code sample in Javascript.

```
const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const multiChannelAdaptorContext = await uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
 const phoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE');
 const request = uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent');
 // You should set correct appClassification here. eg: ORA_SERVICE for Service App
 // Refer to Application Classification Code for the list of supported app classifications
 request.setAppClassification('appClassfication');
 request.setEventId('1'); // Set the event ID here
 const inData = request.getInData();
 phoneContext.publish(request).then((response) => {
 const startCommData = response.getResponseData();
 const engagementContext = startCommData.getEngagementContext();
 const engagementId = engagementContext.getEngagementId();
 const channel = await engagementContext.getChannel(); // 'PHONE'
 })
```

publish

Use this function to publish an operation at engagement Context.

The following code sample shows the syntax:

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

ISubscriptionContext

The ISubscriptionContext object contains the reference to the object on which we've added a subscription on while calling a subscribe or subscribeonce API to listen to application level or object level events. You can call the dispose API on top of this object to dispose or unregister that particular subscription at any point of time.

Functions

dispose

Use this API to dispose or unregister any subscriptions that have been added for an event. By disposing the subscription, the API will not trigger any further notifications to the client side receiver.

```
dispose:() => void;
```

The following code snippet shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

```
const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
const subscriptionContext: ISubscriptionContext = globalContext.subscribe(requestObject, (response:
IEventResponse) => {
// custom code
});
subscriptionContext.dispose();
```

The following code snippet shows an example in JavaScript:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
const subscriptionContext = globalContext.subscribe(requestObject, (response) => {
// custom code
});
subscriptionContext.dispose();
```

iRequestHelper

IRequestHelper has the following functions:

createSubscriptionRequest

This function is used to create requestObject for event subscription.

Here are the parameters:

Parameter	Required?	Description
eventName	Yes	Event name to be subscribed to.

The following code sample shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
const subscriptionContext: ISubscriptionContext = globalContext.subscribe(requestObject, (response:
IEventResponse) => {
// custom code
});
```

The following code sample shows an example in Javascript:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');

globalContext.subscribe(requestObject,(response) => {
```

```
// custom code
});
```

createPublishRequest

This function is used to create requestObject for event subscription.

Here are the parameters:

Parameter	Required?	Description
operationName	Yes	Operation name to be subscribed to.

The following code sample shows an example in Typescript:

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

 const frameworkProvider: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusCustomEventOperation');
 requestObject.setCustomEventName('someName');
 globalContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 //custom code
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in Javascript to generate request object for performing an operation.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusCustomEventOperation');
 requestObject.setCustomEventName('someName');
 globalContext.publish(requestObject).then((message) => {
 //custom code
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

IEventRequest

IEventRequest uses the getEventName function.

The function gets the event name from an event request object created.

The following code sample shows the syntax for getEventName.

```
getEventName(): string;
```

The following code sample shows an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(requestObject: IEventRequest,(response: IEventResponse) => {
 // custom code
 });
 const eventName: string = requestObject.getEventName(); // 'cxEventBusTabOpenEvent'
```

The following code sample shows an example in Javascript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(requestObject,(response) => {
 // custom code
 });
 const eventName = requestObject.getEventName(); // 'cxEventBusTabOpenEvent'
```

IOperationRequest

IOperationRequest uses the getEventName function.

The function gets the operation name from an operation request object created.

The following code sample shows the syntax:

```
getOperationName(): string;
```

The following code sample shows an example in Typescript:

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

 const frameworkProvider: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
 recordContext.publish(requestObject).then((message) => {
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
 const operationName: string = requestObject.getOperationName(); // 'cxEventBusSaveRecordOperation'
```

The following code sample shows an example in Javascript:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
```

```
const requestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
recordContext.publish(requestObject).then((message) => {
// custom code
}).catch((error) => {
// custom code
});
const operationName = requestObject.getOperationName(); // 'cxEventBusSaveRecordOperation'
```

IFieldValueChangeEventRequest

This object is passed as the request object for field value change event subscription.

Functions

setFields

Use this function to set fields of FieldValueChange event request object. See the Field Name Mapping section for the valid field names to be passed.

> **Note:** Currently, complex field aren't supported

The following block shows the syntax for setFields.

```
setFields: (fields: string[]) => void
```

Here are the parameters:

Parameter	Required?	Description
fields	Yes	Field values on top of which subscription needs to be added to.

The following code sample shows an example in Typescript for subscribing to FieldValueChange event where setFields method is used:

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
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;
// custom code
```

```
});
```

The following code sample shows an example in Javascript for subscribing to FieldValueChange event event where setFields method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 // custom code
 });
```

getEventName

This function gets the event name from an event request object created.

The following code sample shows the syntax for getEventName:

```
getEventName(): string;
```

The following code sample shows an example in typescript for subscribing to FieldValueChange event where getEventName method is used:

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 // custom code
 });
 const eventName: string = requestObject.getEventName(); // 'cxEventBusFieldValueChangeEvent'
```

The following code snippet shows an example in Javascript for subscribing to FieldValueChange event where getEventName method is used:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 // custom code
 });
```

```
const eventName = requestObject.getEventName(); // 'cxEventBusFieldValueChangeEvent'
```

ISetFieldValueOperationRequest

This object is passed as the request object for SetFieldValue operation publish API.

The field function is used.

The following code sample shows the syntax for field collection of request that needs to be furnished for SetFieldValue operation publish API.

```
field(): ISetFieldValueRequest;
```

ISetFieldValueRequest

This object is passed as the field information for SetFieldValue operation publish API.

This object uses the setValue function. The following code sample shows the syntax for setValue function on ISetFieldValueRequest object.

```
setValue: (fieldName: string, value: any) => void;
```

Here are the parameters:

Parameter	Required?	Description
fieldName	Yes	Field name to be updated.
value	Yes	Field value that's updated.

The following code sample shows an example in typescript where setValue method is used in SetFieldValue publish API.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''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');
 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 where setValue method is used in SetFieldValue publish API.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 });
```

IGetFieldValueOperationRequest

This object is passed as the request object for get field value operation publish API.

This function is used to set fields of the FieldValueChange event request object.

This object uses the setFields function. The following code sample shows the syntax for setFields function:

```
setFields: (fields: string[]) => void;
```

Here are the parameters:

Parameters	Required?	Description
fields	Yes	Field names that are fetched.

The following code sample shows an example in typescript for publishing GetFieldValue Operation where setFields method is used to construct request object:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in javascript for publishing GetFieldValue Operation where setFields method is used to construct request object:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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((message) => {
// custom code
}).catch((error) => {
// custom code
});
```

IServiceConnectionRequest

The UI Events Framework supports all Service connection requests listed in the Service tab of the Fusion Application.

The IServiceConnectionRequest object must be passed as the request information for InvokeServiceConnection operation publish API.

Functions

setServiceConnectionId

The following example shows the syntax for setServiceConnectionId function for IServiceConnectionRequest object.

```
setServiceConnectionId: (url: string) => void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
url	Yes	URL for the service connection request.

The following code snippet shows an example in Typescript where setServiceConnectionId function is used for making an InvokeServiceConnection request:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
});
```

The following code snippet shows an example in JavaScript where setServiceConnectionId function is used for making an InvokeServiceConnection request:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();

 const restCallRequest = frameworkProvider.requestHelper.createPublishRequest('InvokeServiceConnection');
 restCallRequest.setServiceConnectionId('contacts/getall_contacts');

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

setParameters

The following example shows the syntax for setServiceConnectionId function for IServiceConnectionRequest object.

```
setParameters: (parameter: any) => void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
parameter	Yes	Parameters for the service connection in JSON format.

The following code snippet shows an example in Typescript where setParameters function is used for making an InvokeServiceConnection request:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

 const restCallRequest: IServiceConnectionRequest =
 (frameworkProvider.requestHelper.createPublishRequest('InvokeServiceConnection') as
 IServiceConnectionRequest);
 restCallRequest.setServiceConnectionId('contacts/getall_contacts');
 restCallRequest.setParameters({ 'fields': 'ContactName,MobileNumber,PartyNumber', 'q':
 'MobileNumber=12345' });

 globalContext.publish(restCallRequest).then((message: IOperationResponse) => {
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code snippet shows an example in JavaScript where setParameters function is used for making an InvokeServiceConnection request:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();

 const restCallRequest = frameworkProvider.requestHelper.createPublishRequest('InvokeServiceConnection');
 restCallRequest.setServiceConnectionId('contacts/getall_contacts');
 restCallRequest.setParameters({'fields': 'ContactName,MobileNumber,PartyNumber', 'q':
 'MobileNumber=12345' });

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

setBody

The following example shows the syntax forsetBody function for IServiceConnectionRequest object.

```
setBody: (body: any) => void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
body	Yes	Service connection request body in JSON format.

The following code snippet shows an example in Typescript where setBody function is used for making an InvokeServiceConnection request:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code snippet shows an example in JavaScript where setBody function is used for making an InvokeServiceConnection request:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 }).catch((error) => {
 // custom code
 });
```

IPopFlowRequest

This object must to be passed as the request object for the pop operation Publish API to open any page.

Functions

setOpenPageInNewBrowserTab

This function is used to set whether page need to open in new browser tab or not.

The following example shows the syntax:

```
setOpenPageInNewBrowserTab(openInNewTab: boolean): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
openInNewTab	No	Set to True to open the page in new browser tab and False to open in same tab. The default value is False.

setInputParameters

Function to set input parameters to the opening page.

The following example shows the syntax:

```
setInputParameters(parameters: any): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
parameters	No	You can use any object to set input parameter to opening a page. For appUI pages you can use the mobile number of the opening page or contactId to open a page. For an SR page, it can be {view:'detail'} to see the edit page directly and {view:'foldout'} to see the default foldout view. For IPopFlowGenericRequest, It can be {answerId: "10006003"} to open the article page of Id 10006003.

IPopFlowInAppRequest

This object needs to be passed as the request object for pop operation publish api for in app objects like service request, case, contact, lead, account and opportunity.

Functions

setRecordType

This function is used to set type of the entity object. .

The following example shows the syntax:

```
setRecordType(type: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
type	Yes	The entity type of page.

setRecordId

Function set the ID of the record to open the edit page.

The following example shows the syntax:

```
setRecordId(id: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
id	No	ID of the edit page.

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open create a service request page in typeScript :

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open create a service request:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

The following code snippet shows an example in Typescript example for for IPopFlowInAppRequest to open an edit Service request page with view as details view:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open an edit Service request page with view as details view:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open create a Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open create a Case page:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open a create Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open a create Case page:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open an edit Case in new browser tab:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open an edit Case in new browser tab:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open a create a Contact page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open a create a Contact page:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open an edit Contact page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Contact');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open an edit Contact page:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setRecordType('CDRM_123456');
```

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

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open a create a Account page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open a create a Account page:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

The following code snippet shows an example in Typescript example for IPopFlowInAppRequest to open an edit Account page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

The following code snippet shows an example in JavaScript example for IPopFlowInAppRequest to open an edit Account page

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

IPopFlowAppUIRequest

This object must be passed as the request object for pop operation publish API to open Application UI pages.

Functions

setFlow

This function is used to set flow of application. .

The following example shows the syntax:

```
setFlow(flow: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
flow	No	Flow name of the page.

setPage

Function set to set the page of the application.

The following example shows the syntax:

```
setPage(page: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
page	No	The name of the page.

The following code snippet shows an example in Typescript:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowAppUIRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowAppUIRequest;
 requestObject.setFlow('ec');
 requestObject.setPage('edit');
 requestObject.setInputParameters({contactId: 'C242'});
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

The following code snippet shows an example in JavaScript

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
```

```
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setFlow('ec');
requestObject.setPage('edit');
requestObject.setInputParameters({contactId: 'C242'});
requestObject.setOpenPageInNewBrowserTab(true);
const response = await globalContext.publish(requestObject);
```

IPopFlowGenericRequest

This object must be passed as the request object for pop operation to open any page.

Functions

setFlow

This function is used to set flow of application. .

The following example shows the syntax:

```
setFlow(flow: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
flow	No	Flow URL of the page.

setPage

Function set to set the page of the application.

The following example shows the syntax:

```
setPage(page: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
page	No	The name of the page.

The following code snippet shows an example in Typescript:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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"});
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

The following code snippet shows an example in JavaScript

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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"});
 const response = await globalContext.publish(requestObject);
```

IPopFlowUrlRequest

This object must be passed as the request object for pop operation publish API to open any page by giving URL.

Functions

setUrl

This function is used to set the URL.

The following example shows the syntax:

```
setUrl(url: string): void;
```

The following table shows the parameter:

Parameters

Parameter Name	Required?	Description
url	Yes	Full URL of the page.

IPopFlowResponse

This is the response from Pop Flow operation. It extends the IOperationResponse object.

Functions

getResponseData

This function is used to get the response data of from the Pop Flow operation. The Tab Context of the newly opened page will get back in the application. You can perform all operations and subscriptions in the TabContext that you get in the response.

.

The following example shows the syntax:

```
getResponseData(): ITabContext;
```

The following code snippet shows an example in Typescript of the response from the pop flow operation:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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();
```

The following code snippet shows an example in JavaScript of the response from the pop flow operation:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

IObjectContext

Every event or operation response will have a getContext() method which will return IObjectContext object.

Functions

getObjectType

This function is used to get the object type from the response's context information.

The following code sample shows the syntax for getObjectType:

```
getObjectType(): string;
```

The following code snippet shows an example in Typescript for subscribing to ContextOpen event where getObjectType method is used.

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

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
tabContext.subscribe(requestObject, (response: IEventResponse) => {
console.log(response.getContext().getObjectType()); // usage of getObjectType
});
```

The following code snippet shows an example in javascript for subscribing to ContextOpen event where getObjectType method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getObjectType()); // usage of getObjectType
 });
```

getObjectId

This function is used to get the object unique identifier from the response's context information.

The following code sample shows the syntax for getObjectId.

```
getObjectId(): string;
```

The following code sample shows an example in typescript for subscribing to ContextOpen event where getObjectId method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext().getObjectId()); // usage of getObjectId
 });
```

The following code sample shows an example in Javascript for subscribing to ContextOpen event where getObjectId method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getObjectId()); // usage of getObjectId
 });
```

getTabId

This function is used to get the browser tab's unique identifier from the response object's context information..

The following code sample shows the syntax for getTabId.

```
getTabId(): string;
```

The following code sample shows an example in Typescript for subscribing to ContextClose event where getTabId method is used:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext().getTabId()); // usage of getTabId
 });
```

The following code sample shows an example in Javascript for subscribing to ContextClose event where getTabId method is used:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getTabId()); // usage of getTabId
 });
```

getMsiTabId

This function is used to get the MSI tab's unique identifier from the response object's context information.

The following code sample shows the syntax for getMsiTabId.

```
getMsiTabId(): string;
```

The following code sample shows an example in Typescript for subscribing to ContextClose event where getMsiTabId method is used:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext().getMsiTabId()); // usage of getMsiTabId
 });
```

The following code sample shows an example in Javascript for subscribing to ContextClose event where getMsiTabId method is used:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getMsiTabId()); // usage of getMsiTabId
 });
```

getMsiSubTabId

This function is used to get the MSI subtab's unique identifier from the response's context information. .

The following code sample shows the syntax for getMsiSubTabId.

```
getMsiSubTabId(): string;
```

The following code sample shows an example in Typescript for subscribing to ContextClose event where getMsiSubTabId method is used:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext().getMsiSubTabId()); // usage of getMsiSubTabId
 });
```

The following code sample shows an example in Javascript for subscribing to ContextClose event where getMsiSubTabId method is used:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 consol
```

IExtensionResponse

IExtensionResponse is the generic response object for every event/operation request.

IExtrensionResponse is the generic response object for every event/operation request.

The following code sample shows the syntax for getContext method from event's or operation's response.

```
getContext(): IObjectContext;
```

The following code sample shows an example in typescript for subscribing to ContextOpen event where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject,(response: IEventResponse) => {
 console.log(response.getContext()); // usage of getContext
 });
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getObjectType()); // usage of getContext
 });
```

IEventResponse

IEventResponse is the generic response object for every event subscription.

getEventName

Use this function to get the eventName from the response object.

The following code sample shows the syntax:

```
getEventName(): string;
```

The following code sample shows an example in Typescript for for subscribing to ContextClose event where getEventName method is used..

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject,(response: IEventResponse) => {
 console.log(response.getEventName()); // usage of getEventName
 });
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject,(response) => {
 console.log(response.getEventName()); // usage of getEventName
 });
```

getContext

Use this function to get the context of the response object.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for subscribing to ContextOpen event where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext()); // usage of getContext
 });
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext().getObjectType()); // usage of getContext
 });
```

IOperationResponse

IOperationResponse is the generic response object for every operation publish request.

Functions

getOperationName

Use this function to get the operation name from the response object.

The following code sample shows the syntax for getOperationName method.

```
getOperationName(): string;
```

The following code sample shows an example in Typescript for publishing SaveRecord event where getOperationName method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 console.log(response.getOperationName()); // usage of getOperationName
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for publishing SaveRecord event where getOperationName method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((message) => {
 console.log(response.getOperationName()); // usage of getOperationName
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

getContext

Use this function to get the context of the response object.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for publishing SaveRecord event where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for publishing SaveRecord event where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((message) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

IErrorData

IErrorData is the generic error object for any type of errors in operation failures.

Functions

getStatus

The following code sample shows the syntax of the getStatus function.

```
getStatus: () => string;
```

The following code sample shows an example in Typescript for publishing SaveRecord event where IErrorData's getStatus method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 console.log(message);
 }).catch((error: IErrorData) => {
 console.log(error.getStatus()); // usage of getStatus()
 });
```

The following code snippet shows an example in JavaScript for publishing SaveRecord event where IErrorData's getStatus method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((message) => {
 console.log(message);
 }).catch((error) => {
 console.log(error.getStatus()); // IErrorData getStatus() usage
 });
```

getMessage

The following code sample shows the syntax of getMessage function.

```
getMessage: () => string;
```

The following code sample shows an example in Typescript for publishing SaveRecord event where IErrorData's getMessage method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
```

```
recordContext.publish(requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest).then((message:
CX_SVC_UI_EVENTS_FRAMEWORK.IOperationResponse) => {
console.log(message);
})
.catch((error: CX_SVC_UI_EVENTS_FRAMEWORK.IErrorData) => {
console.log(error.getMessage()); // IErrorData getMessage() usage
});
```

The following code sample shows an example in JavaScript for publishing SaveRecord event where IErrorData's getMessage method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((message) => {
 console.log(message);
 }).catch((error) => {
 console.log(error.getMessage()); // IErrorData getMessage() usage
 });
```

IFieldValueChangeData

Use to get field name of the updated field in FieldValueChange event subscription response.

Functions

getFieldName

Use this function to get field name of the updated field in FieldValueChange event subscription response.

The following code sample shows the syntax for getFieldName method.

```
getFieldName: () => string;
```

The following code sample shows an example in Typescript for subscribing to FieldValueChange event where getFieldName method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 console.log(response.getResponseData().getFieldName());
 });
```

The following code sample shows an example in JavaScript for subscribing to FieldvalueChange event where getFieldName method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 console.log(response.getResponseData().getFieldName());
 });
```

getOldValue

Use this function to get old value of the updated field in FieldValueChange event subscription response.

The following code sample shows the syntax for getFieldName method.

```
getOldValue: () => string | number | boolean;
```

The following code sample shows an example in Typescript for subscribing to FieldValueChange event where getOldValue method: is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 console.log(response.getResponseData().getOldValue());
 });
```

The following code sample shows an example in JavaScript for subscribing to FieldValueChange event where getOldValue method: is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 console.log(response.getResponseData().getOldValue());
 });
```

getNewValue

Use this function to get new value of the updated field in FieldValueChange event subscription response.

The following code sample shows the syntax for getNewValue method.

```
getNewValue: () => string | number | boolean;
```

The following code sample shows an example in Typescript for subscribing to FieldvalueChange event where getNewValue method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 console.log(response.getResponseData().getNewValue());
 });
```

The following code sample shows an example in JavaScript for subscribing to FieldvalueChange event where getNewValue method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 console.log(response.getResponseData().getNewValue());
 });
```

IFieldValueCollection

This field will contain the collection of fields that we get in GetFieldValue operation response.It uses the getField method.

The getField method will fetch a particular field's data in the GetFieldValue operation response.

The following code sample shows the syntax for getField method.

```
getField: (fieldName: string) => IFieldData;
```

Parameters

Here's the parameter for this method:

Parameter	Required?	Description
fieldName	Yes	Name of a particular field of which data we need to retrieve from the GetFieldValue Operation Response

The following code sample shows an example in Typescript for publishing GetFieldValue Operation where getField method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 console.log(response.getResponseData().getField('ServiceRequest.Title')) // usage of getField method
 })
 .catch((error: IErrorData) => {
 // error
 });
```

The following code sample shows an example in JavaScript for publishing GetFieldValue Operation where getFieldName method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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((message) => {
 console.log(response.getResponseData().getField('ServiceRequest.Title')) // usage of getField method
 })
 .catch((error) => {
 // error
 });
```

IFieldData

This object will contain a field's information, like fieldName and its value. This object is received as the response of getFieldValue operation response's getField method.

getValue

The following code sample shows the syntax for getting value of a field in getFieldValue operation's response:

```
getValue: () => string;
```

The following code sample shows an example in typescript for publishing GetFieldValue Operation where getValue method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 // usage of getValue method
 console.log(response.getResponseData().getField('ServiceRequest.Title').getValue())
 })
 .catch((error: IErrorData) => {
 // error
 });
```

The following code sample shows an example in javascript for publishing GetFieldValue Operation where getValue method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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((message) => {
 // usage of getValue method
 console.log(response.getResponseData().getField('ServiceRequest.Title').getValue())
 })
 .catch((error) => {
 // error
 });
```

getFieldName

The following code sample shows the syntax for getting field name of a field getFieldValue operation's response:

```
getFieldName: () => string;
```

The following code sample shows an example in typescript for publishing GetFieldValue Operation where getFieldName method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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;
 console.log(response.getResponseData().getField('ServiceRequest.Title').getFieldName()) // usage of
 getFieldName method
 })
 .catch((error: IErrorData) => {
 // error
```

```
});
```

The following code sample shows an example in javascript for publishing GetFieldValue Operation where getFieldName method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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((message) => {
 // usage of getFieldName method
 console.log(response.getResponseData().getField('ServiceRequest.Title').getFieldName())
 })
 .catch((error) => {
 // error
 });
```

ITabEventResponse

This object returns the response of the TabOpen event subscription.

Functions

getResponseData

Use this function to get Response data from the TabOpen event subscription response.

Here's the syntax:

```
getResponseData(): ITabContext;
```

getEventName

Use this function to get event name data from the TabOpen event subscription response.

Here's the syntax:

```
getEventName(): string;
```

getContext

Use this function to get event name data from the TabOpen event subscription response.

Here's the syntax:

```
getEventName(): string;
```

getType

Use this function to get type from the TabOpen event subscription response. The type shows the type of newly opened tab as browser tab, MSI tab or MSI sub tab.

Here's the syntax:

```
getType(): string;
```

The following code sample shows an example in Typescript for for subscribing to TabOpen event.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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: string = tabOpenResponse.getType();
 })
```

The following code sample shows an example in JavaScript for for subscribing to TabOpen event.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await getGlobalContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(payload, (response) => {
 const tabContext = response.getResponseData();
 const type = tabOpenResponse.getType();
 })
```

ITabCloseEventResponse

This object returns the response of the TabClose event subscription.

Functions

getResponseData

Use this function to get Response data from the TabOpen event subscription response.

Here's the syntax:

```
getResponseData(): ITabContext;
```

getEventName

Use this function to get event name data from the TabOpen event subscription response.

Here's the syntax:

```
getEventName(): string;
```

getContext

Use this function to get event name data from the TabOpen event subscription response.

Here's the syntax:

```
getEventName(): string;
```

The following code sample shows an example in Typescript for subscribing to TabClose event.

```
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()); // Close Tab's tab information
 console.log(responseData.getEventName()); // 'cxEventBusTabCloseEvent'
 console.log(responseData.getContext()); // contextObject info
 });
 }
```

The following code sample shows an example in JavaScript for subscribing to TabClose event.

```
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()); // Closed Tab's tab information
 console.log(responseData.getEventName()); // 'cxEventBusTabCloseEvent'
 console.log(responseData.getContext()); // contextObject info
 });
 }
```

ITabInfo

Functions

getTabId

Use this function to to get the browser tabId value from the TabClose event subscription response.

Here's the syntax:

```
getTabId(): string;
```

getMsiTabId

Use this function to get the MSI tab ID from the TabClose event subscription response.

Here's the syntax:

```
getMsiTabId(): string;
```

getMsiSubTabId

Use this function to get the MSI sub tab ID from the TabClose event subscription response.

Here's the syntax:

```
getMsiSubTabId(): string;
```

The following code sample shows an example in Typescript for subscribing to TabClose Event where ITabInfo object is used..

```
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()); // Closed Tab's browser tab identifier
 console.log(responseData.getResponseData().getMsiTabId()); // Closed Tab's msi tab identifier
 console.log(responseData.getResponseData().getMsiSubTabId()); // Closed Tab's msi sub tab identifier
 });
 }
```

The following code sample shows an example in JavaScript for for subscribing to TabClose event.

```
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()); // Closed Tab's browser tab identifier
 console.log(responseData.getResponseData().getMsiTabId()); // Closed Tab's msi tab identifier
 console.log(responseData.getResponseData().getMsiSubTabId()); // Closed Tab's msi sub tab identifier
 });
 }
```

IEngagementInfo

This object holds information about an engagement associated with a TabContext.

Functions

getEngagementId

Use this function to get engagement ID of an engagement..

Here's the syntax:

```
getEngagementId(): string;
```

getEngagementType

Use this function to get the engagement type of an engagement.

Here's the syntax:

```
getEngagementType(): string;
```

getInteractionId

Use this function to get the interaction ID of an engagement.

Here's the syntax:

```
getInteractionId(): string;
```

getWrapUpId

Use this function to get the wrap up ID of an engagement.

Here's the syntax:

```
getWrapUpId(): string;
```

getInData

Use this function to get in data of an engagement.

Here's the syntax:

```
getInData(): Record<any, any>;
```

getOutData

Use this function to get out data of an engagement.

Here's the syntax:

```
getOutData(): Record<any, any>;
```

isActiveEngagement

Use this function to return information on whether an engagement is active or not.

Here's the syntax:

```
isActiveEngagement(): boolean;
```

The following code sample shows an example in Typescript for getting the engagement information for current browser tab context.

```
let uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
let frameworkProvider: IUiEventsFrameworkProvider = await uefProvider.uiEventsFramework.initialize('My
 App');
let tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
let engagementInfo: IEngagementInfo = await tabContext.getEngagementInfo();
console.log(engagementInfo.getEngagementId());
console.log(engagementInfo.getEngagementType());
console.log(engagementInfo.getInteractionId());
console.log(engagementInfo.getWrapUpId());
console.log(engagementInfo.getInData());
console.log(engagementInfo.getOutData());
console.log(engagementInfo.isActiveEngagement());
```

The following code sample shows an example in JavaScript for getting the engagement information for current browser tab context.

```
let uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
let frameworkProvider = await uefProvider.uiEventsFramework.initialize('My App');
let tabContext = await frameworkProvider.getCurrentBrowserTabContext();
let engagementInfo = await tabContext.getEngagementInfo();
console.log(engagementInfo.getEngagementId());
console.log(engagementInfo.getEngagementType());
console.log(engagementInfo.getInteractionId());
console.log(engagementInfo.getWrapUpId());
console.log(engagementInfo.getInData());
console.log(engagementInfo.getOutData());
console.log(engagementInfo.isActiveEngagement());
```

ITabChangeEventResponse

This object returns the response of the TabChange event subscription.

Functions

getResponseData

Use this function to get Response data from the TabChange event subscription response.

Here's the syntax:

```
getResponseData(): ITabChangeResponse;
```

getEventName

Use this function to get the origin type of the tab change event. The tab change event origin could be from a browser tab, MSI tab or from an MSI-sub tab. .

Here's the syntax:

```
getType(): string;
```

The following code sample shows an example in Typescript for getting the engagement information for current browser tab context.

```
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();
 });
```

The following code sample shows an example in JavaScript for getting the engagement information for current browser tab context.

```
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();
 });
```

ITabChangeResponse

This object returns the current tab's context and the previous tab's context on a tab change event.

Functions

getCurrentTab

Use this function to get the current tab's context from the TabChange event subscription response.

Here's the syntax:

```
getCurrentTab(): ITabContext;
```

getPreviousTab

Use this function to get the previous tab's context from the TabChange event subscription response.

Here's the syntax:

```
getPreviousTab(): ITabContext;
```

The following code sample shows an example in Typescript for subscribing to a TabChange event.

```
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();
});
```

The following code sample shows an example in JavaScript for subscribing to a TabChange event.

```
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) => {
const type: string = response.getType();
const tabChangeResponse = response.getResponseData().
const currentTab = tabChangeResponse.getCurrentTab();
const previousTab = tabChangeResponse.getPreviousTab();
});
```

IFocusTabResponseData

This is the response for the Tab Focus operation. It extends the IOperationResponse object.

Functions

getResponseData

Use this function to get the response data of Tab Focus operation.

Here's the syntax:

```
getResponseData(): ITabChangeResponse;
```

The following code sample shows an example in Typescript of the Tab Focus operation.

```
const publishFocusTab = async () => {
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
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) => {
});
};
```

The following code sample shows an example in JavaScript of the Tab Focus operation.

```
const publishFocusTab = async () => {
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
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) => {
});
};
```

getContext

Use this function to get the context of the response object..

Here's the syntax:

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for publishing to FocusTab operation where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const context: IObjectContext = (message as IFocusTabResponseData).getContext();
 }).catch((error: IErrorData) => {
 // error
 });
```

The following code sample shows an example in JavaScript for publishing to FocusTab operation where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((response) => {
 const context = message.getContext();
 }).catch((error) => {
 // error
 });
```

ITabCloseOperationResponse

This is the response for the Tab Close operation. It extends the IOperationResponse object.

Functions

getResponseData

Use this function to get the response data of Tab Close operation.

Here's the syntax:

```
getResponseData(): ITabInfo;
```

The following code sample shows an example in Typescript of the Tab Closeoperation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 // error
 });
```

The following code sample shows an example in JavaScript of the Tab Close operation.

```
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) => {
 // error
 });
```

getContext

Use this function to get the context of the response object.

Here's the syntax:

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for publishing to the TabClose operation where the getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusCloseTabOperation');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const context: IObjectContext = message.getContext();
 }).catch((error: IErrorData) => {
 });
```

The following code sample shows an example in JavaScript for publishing to the TabClose operation where the getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const payload = frameworkProvider.requestHelper.createPublishRequest('cxEventBusCloseTabOperation');
 tabContext.publish(payload).then((response) => {
 const context = message.getContext();
 }).catch((error) => {
```

```
});
```

IContextOpenEventResponse

IContextOpenEventResponse is the response object for ContextOpenEvent subscription.

Functions

getResponseData

Use this function to get response data for for ContextOpenEvent subscription.

Here's the syntax:

```
getResponseData(): IRecordContext;
```

The following code sample shows an example in Typescript for subscribing to ContextOpen event where getResponseData method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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 where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
 tabContext.subscribe(requestObject, (message: IEventResponse) => {
 const recordContext: IRecordContext = message.getResponseData();
 })
```

getContext

Use this function to get the context of the response object.

Here's the syntax:

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for subscribing to ContextOpen event where getContext method is used.

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

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
tabContext.subscribe(requestObject, (response: IEventResponse) => {
console.log(response.getContext()); // usage of getContext
});
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event where getContext method is used.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext = await frameworkProvider.getTabContext();
const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent')
tabContext.subscribe(requestObject, (response) => {
console.log(response.getContext()); // usage of getContext
});
```

IContextResponse

The response object for events that provide information only about the context. For example, ContextCloseEvent subscription.

Functions

getResponseData

Use this function to get response data for Context-related event subscriptions.

Here's the syntax:

```
getResponseData(): IObjectContext;
```

The following code sample shows an example in Typescript for subscribing to the ContextClose event where the getResponseData method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
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());
console.log(context.getTabId());
});
```

The following code sample shows an example in JavaScript for subscribing to the ContextClose event where the getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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());
 });
```

getContext

Use this function to get the context of the response object.

Here's the syntax:

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for subscribing to the ContextClose event where the getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response: IEventResponse) => {
 console.log(response.getContext()); // usage of getContext
 });
```

The following code sample shows an example in JavaScript for subscribing to the ContextClose event where the getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent')
 tabContext.subscribe(requestObject, (response) => {
 console.log(response.getContext()); // usage of getContext
 });
```

IGetAgentInfoResponse

The response object while retrieving logged-in agent's information.

Functions

getFirstName

Use this function to get the first name of logged in agent.

Here's the syntax:

```
getFirstName(): string;
```

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

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 });
```

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''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 globalContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
 });
```

getLastName

Use this function to get the last name of logged in agent.

Here's the syntax:

```
getLastName(): string;
```

The following code sample shows an example in Typescript for getting agent's last name using getLastName method.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 });
```

The following code sample shows an example in JavaScript for getting agent's last name using getLastName method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
```

```
globalContext.publish(requestObject).then((response) => {
console.log(response.getLastName());
});
```

getEmailAddress

Use this function to get the email address of signed in agent.

Here's the syntax:

```
getEmailAddress(): string;
```

The following code sample shows an example in Typescript for getting agent's email address using getEmailAddress method.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 });
```

The following code sample shows an example in JavaScript for getting agent's email address using getEmailAddress method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 globalContext.publish(requestObject).then((response) => {
 console.log(response.getEmailAddress());
 });
```

getUserName

Use this function to get the user name of the signed in agent.

Here's the syntax:

```
getUserName(): string;
```

The following code sample shows an example in Typescript for getting the agent's user name using getUserName method.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
```

```
});
```

The following code sample shows an example in JavaScript for getting the agent's user name using getUserName method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 globalContext.publish(requestObject).then((response) => {
 console.log(response.getUserName());
 });
```

getPartyId

Use this function to get the party ID of the signed in agent.

Here's the syntax:

```
getUserName(): string;
```

The following code sample shows an example in Typescript for getting agent's party ID using getPartyId method.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
 });
```

The following code sample shows an example in JavaScript for getting agent's party ID using getPartyId method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 globalContext.publish(requestObject).then((response) => {
 console.log(response.getPartyId());
 });
```

IGetFieldsInUIResponse

The response object of Get Fields in UI operation

Here's the syntax:

```
export interface IGetFieldsInUIResponse extends IOperationResponse {
 getResponseData(): IGetFieldsInUIResponseData;
}
```

IGetFieldsInUIResponseData

IGetFieldsInUIResponseData has the getFieldsList method exposed which provides a list of fields rendered in the UI.

Here's the syntax:

```
export interface IGetFieldsInUIResponseData {
 getFieldsList: () => string[];
}
```

IGetFieldValueResponse

IGetFieldValueResponse is the response object for GetFieldValue operation response.

Functions

getResponseData

Use this function to get response data for for GetFieldValue operation response.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IFieldValueCollection;
```

The following code sample shows an example in typescript for subscribing to FieldValue operation where getResponseData method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const getfieldValuePayload: IGetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
 IGetFieldValueOperationRequest);

 getfieldValuePayload.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);
 recordContext.publish(getfieldValuePayload).then((message) => {
 const response = message as IGetFieldValueResponse;
 console.log(.getResponseData().getField('ServiceRequest.Title').getValue());
 }).catch((error: IErrorData) => {
 error.getMessage();
 });
```

The following code sample shows an example in javascript for subscribing to FieldValue operation where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation').setFields(['ServiceRequest

 recordContext.publish(requestObject).then((response) => {
 console.log(response.getResponseData().getField('ServiceRequest.Title').getValue());
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

getContext

Use this function to get the context of the response's context.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in typescript where getContext method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const getfieldValuePayload: IGetFieldValueOperationRequest = (frameworkProvider.requestHelper.
 createPublishRequest('cxEventBusGetFieldValueOperation') as IGetFieldValueOperationRequest);
 getfieldValuePayload.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

 recordContext.publish(getfieldValuePayload).then((message) => {
 const response = message as IGetFieldValueResponse;
 console.log(response.getContext()); // usage of getContext
 }).catch((error: IErrorData) => {
 error.getMessage();
 });
```

The following code sample shows an example in javascript where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const getfieldValuePayload =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation').setFields(['ServiceRequest

 recordContext.publish(getfieldValuePayload).then((response) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

IServiceConnectionResponse

The response object for InvokeServiceConnection operation response.

Functions

getResponseData

Use this function to get response data for for InvokeServiceConnection operation response.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IServiceConnectionResponseData;
```

The following code sample shows an example in typescript for subscribing to InvokeServiceConnection operation where getResponseData method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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: IServiceConnectionResponse) => {
 // custom code
 const response = message.getResponseData();
 console.log(response.getStatus());
 console.log(response.getBody());
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in javascript for subscribing to InvokeServiceConnection operation where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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
});
```

IOperationSuccessData

This object provides the response message for an operation's success response.

The following code sample shows the syntax for IOperationSuccessData

```
getMessage: () => string;
```

ISetFieldValueResponse

ISetFieldValueResponse is the response object for SetFieldValue operation response.

getResponseData

Use this function to get response data for for Set FieldValue subscription.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IOperationSuccessData;
```

The following code sample shows an example in typescript for subscribing to Set FieldValue operation where getResponseData method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 let setFieldValuePayloadData: ISetFieldValueOperationRequest;
 setFieldValuePayloadData =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest;
 setFieldValuePayloadData.field().setField('ServiceRequest.Title').setValue('SR101');

 recordContext.publish(setFieldValuePayloadData).then((response) => {
 const responsePayload = response as ISetFieldValueResponse;
 console.log(responsePayload.getResponseData().getMessage());
 }
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for subscribing to Set FieldValue operation where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
```

```
setFieldValuePayloadData =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
setFieldValuePayloadData.field().setField('ServiceRequest.Title').setValue('SR101');
recordContext.publish(setFieldValuePayloadData).then((response) => {
console.log (response.getResponseData().getMessage());
}).catch((error) => {
console.log(error.getMessage());
});
```

getContext

Use this function to get the context of the response's context object.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in typescript where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider =
 await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const getfieldValuePayload: IGetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
 IGetFieldValueOperationRequest);
 getfieldValuePayload.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);
 recordContext.publish(getfieldValuePayload).then((message) => {
 const response = message as ISetFieldValueResponse;
 console.log(response.getContext()); // usage of getContext
 }).catch((error: IErrorData) => {
 error.getMessage();
 });
```

The following code sample shows an example in JavaScript where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 setFieldValuePayloadData =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 setFieldValuePayloadData.field().setField('ServiceRequest.Title').setValue('SR1');
 recordContext.publish(getfieldValuePayload).then((response) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

IOnSaveExtensionContext

This object provides the objectId, oldObjectId and Object type from the response object of OnAfterSave event.

getObjectId

The following code sample shows the syntax for getting the objectId:

```
getObjectId: () => string;
```

getOldObjectId

The following code sample shows the syntax for getting the old objectId of the record. The old objectId should be minus number if the event is create and should be same if the event is save.

```
getOldObjectId: () => string;
```

getObjectType

The following code sample shows the syntax for getting the objectType:

```
getObjectType: () => string;
```

IOnAfterSaveEventResponse

IOnAfterSaveResponse is the response object for OnAfterSave event.

Functions

getReponseData

Use this function to get response data of OnAfterSave event's subscription.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IOnAfterExtensionContext;
```

The following code sample shows an example in Typescript for subscribing to OnAfterSaveEvent where getResponseData method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (message) => {
 const response = message as CX_SVC_UI_EVENTS_FRAMEWORK.IOnAfterSaveEventResponse;
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getOldObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 });
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getOldObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 });
```

getContext

Use this function to get the context of the response object.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript where getContext method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: CX_SVC_UI_EVENTS_FRAMEWORK.IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (message) => {
 const response = message as IOnAfterSaveEventResponse;
 console.log(response.getContext()); // usage of getContext
 });
```

The following code sample shows an example in JavaScript where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (response) => {
 console.log(response.getContext()); // usage of getContext
 });
```

IOnBeforeSaveEventResponse

IOnBeforeSaveResponse is the response object for OnBeforeSave event.

Functions

getResponseData

Use this function to get response data of onBeforeSave event subscription.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IObjectContext;
```

The following shows an example in Typescript for subscribing to OnBeforeSave where getResponseData method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');
 recordContext.subscribe(requestObject, (message) => {
 const response = message as IOnBeforeSaveEventResponse;
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 });
```

The following code sample shows an example in JavaScript for subscribing to OnBeforeSave event where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');
 recordContext.subscribe(requestObject, (response) => {
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 });
```

getContext

Use this function to get the context of the response's context object.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: IEventRequest =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');
 recordContext.subscribe(requestObject, (message) => {
```

```
const response = message as IOnAfterSaveEventResponse;
console.log(response.getContext()); // usage of getContext
});
```

The following code sample shows an example in JavaScript where getContext method is used.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();

const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');
recordContext.subscribe(requestObject, (response) => {
console.log(response.getContext()); // usage of getContext
});
```

IFieldValueChangeEventResponse

IFieldValueChangeEventResponse is the response object for FieldValueChange event.

Functions

getResponseData

Use this function to get response data of FieldValueChange event subscription.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IFieldValueChangeData;
```

The following code sample shows an example in Typescript for subscribing to FieldValueChange event where getResponseData method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
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;
console.log(response.getResponseData().getFieldName());
console.log(response.getResponseData().getOldValue());
console.log(response.getResponseData().getNewValue());
});
```

The following code sample shows an example in JavaScript for subscribing to FieldValueChange event where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 console.log(response.getResponseData().getFieldName());
 console.log(response.getResponseData().getOldValue());
 console.log(response.getResponseData().getNewValue());
 });
```

getContext

Use this function to get the context of the response object's context.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for subscribing to FieldValueChange event where getContext method is used.

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

 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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 CX_SVC_UI_EVENTS_FRAMEWORK.IFieldValueChangeEventResponse;
 console.log(response.getContext());
 });
```

The following code sample shows an example in JavaScript for subscribing to FieldValueChange event where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 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) => {
 console.log(response.getContext()); // usage of getContext
 });
```

ISaveRecordResponse

ISaveRecordResponse is the response object for saveRecord operation.

Functions

getResponseData

Use this function to get response data of Save record operation.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IOnAfterExtensionContext;
```

The following code sample shows an example in Typescript for saveRecord operation where getResponseData method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

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

 recordContext.publish(requestObject: IOperationRequest ).then((message) => {
 const response = message as ISaveRecordResponse; //custom code
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getOldObjectId()); // usage of getResponseData
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for saveRecord operation where getResponseData method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

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

 recordContext.publish(requestObject).then((response) => {
 console.log(response.getResponseData().getObjectType()); // usage of getResponseData
 console.log(response.getResponseData().getObjectId()); // usage of getResponseData
 console.log(response.getResponseData().getOldObjectId()); // usage of getResponseData
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

getContext

Use this function to get the context of the response object's context.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for SaveRecord Operation where getContext method is used.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider =await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

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

 recordContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 const response = message as ISaveRecordResponsePayload;
 console.log(response.getContext()); // usage of getContext
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for SaveRecord operation where getContext method is used.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((response) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

IUpdateSidePaneRequest

This object must be passed as the request object for update sidePane operation publish API.

Here's the syntax:

```
setVisibility(visibility: boolean): void;
```

Functions

setVisibility

Use this function to set visibility of the sidePane icon:

```
setVisibility(visibility: boolean): void;
```

Parameter

Parameter Name	Required?	Description
visibility	No	Visibility of the Side Pane icon.

The following code sample shows an example in Typescript for updating side pane operation to update section visibility.

```
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(() => { })
 }
```

The following code sample shows an example in JavaScript for updating side pane operation to update section visibility.

```
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(() => { })
 }
```

setSectionId

Use this function to set a different section for a sidePane which is already customized to another section ID.

```
setSectionId(sectionId: string): void;
```

Parameter

Parameter Name	Required?	Description
sectionId	Yes	The section ID.

The following code sample shows an example in Typescript for the side pane operation to update section ID.

```
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(() => { })
}
```

The following code sample shows an example in JavaScript for the side pane operation to update section ID.

```
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(() => { })
 }
```

setIcon

Use this function to set the icon of the sidePane:

```
setIcon(icon: boolean): void;
```

Parameter

Parameter Name	Required?	Description
icon	Yes	The icon symbol of the Side Pane.

The following code sample shows an example in Typescript for updating the side pane operation to update icon.

```
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(() => { })
 }
```

The following code sample shows an example in JavaScript for updating the side pane operation to update icon.

```
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(() => { })
```

```
}
```

getContext

Use this function to get the context of the response object's context.

The following code sample shows the syntax for getContext method.

```
getContext(): IObjectContext;
```

The following code sample shows an example in Typescript for SaveRecord Operation where getContext method is used.

```
/// <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: CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject: IOperationRequest ).then((message: IOperationResponse) => {
 const response = message as ISaveRecordResponsePayload;
 console.log(response.getContext()); // usage of getContext
 }).catch((error: IErrorData) => {
 console.log(error.getMessage());
 });
```

The following code sample shows an example in JavaScript for SaveRecord operation where getContext method is used.

```
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) => {
 console.log(response.getContext()); // usage of getContext
 }).catch((error) => {
 console.log(error.getMessage());
 });
```

ISidePaneOpenEventResponse

Use this function to get a response from the SidePane open event.

Functions

getResponseData

Here's the syntax:

```
getResponseData(): ISidePaneData
```

ISidePaneData

Use this function to get a response for the sidePane open event response.

Functions

getActiveSectionId

Use this function to get the session ID of the opened sidePane:

```
getActiveSessionId(): string;
```

getId

Use this function to get the ID of the opened sidePane:

```
getId(): string;
```

Here's an example in Typescript of SidePaneOpen Event where the ISidePaneOpenEventResponse object is used.

```
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 an example in JavaScript of SidePaneOpen Event where the ISidePaneOpenEventResponse object is used.

```
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());
 });
 }
```

ISidePaneCloseEventResponse

Use this to get the response of SidePane close event.

Functions

getResponseData

Here's the syntax:

```
getResponseData(): ISidePaneData
```

ISidePaneCloseData

Use this function to get a response for the sidePane open event response.

Functions

getId

Use this function to get the ID of the closed sidePane:

```
getId(): string;
```

Here's an example in Typescript of SidePaneOpen event subscription:

```
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 an example in JavaScript of of SidePaneOpen event subscription.

```
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 = res.getResponseData();
 const id = sidePaneCloseData.getId();
 });
 }
```

IOpenModalWindowRequest

Pass this object as the request object for modal operation publish API.

Functions

setId

Use this function to set the Unique ID of the modal. In a success response, you'll get the same Id value:

```
setId(id: boolean): void;
```

Parameter

Parameter Name	Required?	Description
Id	Yes	Unique ID of the modal.

setUrl

Use this function to set the URL of the modal content. This URL render in an iFrame to display in the modal.

```
setURL(url: string): void;
```

Parameter

Parameter Name	Required?	Description
url	Yes	Content of the modal.

setTitle

Use this function to set the title of the modal.

```
setTitle(title: string): void;
```

Parameter

Parameter Name	Required?	Description
Title	No	Title of the modal.

setClosable

Use this function to set the modal closable or not. If it isn't set to True, the Close icon won't be available in the modal. You must close it with the CloseModal operation:

```
setClosable(closable: boolean): void;
```

Parameter

Parameter Name	Required?	Description
closable	No	The default value is False, and the Close icon wont be available in the modal. You need to close it with CloseModal operation.

setStyle

Use this function to set the style of the modal:

```
setStyle(style: any): void;
```

Parameter

Parameter Name	Required?	Description
style	No	The syle of the object. For example: width: 100px, height 100px.

Here's an example in Typescript of the OpenModal action:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://slc05zrk.us.oracle.com:8080/clientAppTest/modalClose.html');
 requestObject.setId('modal_103');
 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 an example in JavaScript of of SidePaneOpen event subscription.

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal')
 requestObject.setURL('https://slc05zrk.us.oracle.com:8080/clientAppTest/modalClose.html');
 requestObject.setId('modal_103');
```

```
requestObject.setTitle('Test title');
requestObject.setClosable(true);
requestObject.setStyle({ width:'1000px', height:'1000px' });
const response = await modalWindowContext.publish(requestObject);
const id = response.getResponseData().getId();
}
```

ICloseModalWindowRequest

This object must be passed as the request object for modal operation publish API.

Functions

setId

Use this function to set the Unique ID of the modal. In a success response, you'll get the same ID value:

```
setId(id: boolean): void;
```

IOpenPopupWindowRequest

This object must be passed as the request object for the pop-up operation publish API.

Functions

setId

Use this function to set the unique ID of the pop-up. In a successful response, you'll get the same ID value:

```
setId(id: boolean): void;
```

Parameter

Parameter Name	Required?	Description
Id	Yes	Unique ID of the modal.

setUrl

Use this function to set the URL of the modal content. This URL render in an iFrame to display in the modal.

```
setURL(url: string): void;
```

Parameter

Parameter Name	Required?	Description
url	Yes	Content of the modal.

setTitle

Use this function to set the title of the modal.

```
setTitle(title: string): void;
```

Parameter

Parameter Name	Required?	Description
Title	No	Title of the modal.

setClosable

Use this function to set the modal closable or not. If it isn't set to True, the Close icon won't be available in the modal. You must close it with the CloseModal operation:

```
setClosable(closable: boolean): void;
```

Parameter

Parameter Name	Required?	Description
closable	No	The default value is False, and the Close icon wont be available in the modal. You need to close it with CloseModal operation.

setStyle

Use this function to set the style of the modal:

```
setStyle(style: any): void;
```

Parameter

Parameter Name	Required?	Description
style	No	The style of the object. For example: width: 100px, height 100px.

ICustomEventSubscriptionRequest

This object must be passed as the request object for subscribe custom event or, subscribe API.

Functions

setCustomEventName

Use this function to set customEventName to which user is interested in adding subscription to:

```
setCustomEventName(eventName: string): void;
```

Parameter

Parameter Name	Required?	Description
eventName	Yes	Any name as customEventName.

ICustomEventSubscriptionResponse

The response object of CustomEvent subscription. It extends the IEventResponse object.

Functions

setResponseData

Use this function to get response data from the custom event subscription response:

```
getResponseData(): ICustomEventSubscriptionResponseData;
```

ICustomEventRequest

This object must be passed as the request object for custom event's publish API.

Functions

setEventPayload

Use this function to set customEventName to which user is interested in invoking publish API:

```
setCustomEventName(eventName: string): void;
```

Parameter

Parameter Name	Required?	Description
eventName	Yes	Any name as customEventName .

setEventPayload

Use this function to set data in the custom event publish request object. This data is sent to the receiver of this publish request:

```
setEventPayload(payload: any): void;
```

Parameter

Parameter Name	Required?	Description
payload	Yes	Any data you want to send along with the custom event request

ICustomEventResponse

The response object of the CustomEvent publish API. It extends the IOperationResponse object.

Functions

The extra functions available in the ICustomEventResponse other than those in IOperationResponse are shown as follows:

getResponseData

Use this function to get response data from the custom event publish response.

```
getResponseData(): ICustomEventResponseData;
```

ICustomEventResponseData

The response object of the getResponseData method of the CustomEvent publish response.

Functions

getData

Use this function to get data from the custom event publish response.

```
getData(): any;
```

getCustomEventName

Use this function to get custom event name for which from the custom event publish api is executed.

```
getCustomEventName(): string;
```

IDataLoadEventResponse

The response object for the DataLoadEvent event.

Function

getReponseData

Use this method to get the response data from IDataLoadEventResponse object.

The following code sample shows the syntax for getResponseData method.

```
getResponseData(): IDataLoadEventData;
```

IDataLoadEventData

Returns data from DataLoad event response.

Function

getCurrentView

Use this method to get the current view of the record to which DataLoad subscription is added to.

```
getCurrentView(): string;
```

INotificationActionPayload

The format of actions provided with setActions.

For example,

```
[{id: 'openPage1', name: 'Open Page 1'}, {id: 'openPage2', name: 'Open Page 2'}]
```

Here's a Typescript example:

```
let actionsList: INotificationActionPayload = [{id: 'openPage1', name: 'Open Page 1'}, {id: 'openPage2',
 name: 'Open Page 2'}];
```

Here's a JavaScript example:

```
let actionsList = [{id: 'openPage1', name: 'Open Page 1'}, {id: 'openPage2', name: 'Open Page 2'}];
```

IShowNotificationRequest

This object needs to be passed as the request object for show notification publish API.

Functions

setTitle

Function to set Title.

```
setTitle(title: string): void;
```

Parameters

Parameter Name	Required?	Description
title	Yes	Title of the notification.

setSummary

Function to set Summary of notification message.

```
setSummary(title: string): void;
```

Parameters

Parameter Name	Required?	Description
summary	Yes	Summary of the notification

setIcon

Icon of the notification as url.

```
setIcon(icon: string): void;
```

Parameters

Parameter Name	Required?	Description
icon	No	The icon of the notification.

setClosable

Function to set manual close functionality.

```
setClosable(isClosable: boolean): void;
```

Parameters

Parameter Name	Required?	Description
isClosable	No	Should display close icon or not in the notification.

setAutoTimeout

Function to set AutoTimeout. For example: 1 for 1 second, 6 for 6 seconds and so on. Notification will close automatically after this given time.

```
setAutoTimeout(time: number): void;
```

Parameters

Parameter Name	Required?	Description
isClosable	No	Should display close icon or not in the notification.

setType

Function to set type. Possible option are: error, warning, confirmation, info and none. Default will be Info.

```
setType(type: string): void;
```

Parameters

Parameter Name	Required?	Description
type	No	The type of the notification. It shows corresponding icon to the given type if Icon isn't provided with setIcon.

setActions

Function to set actions in notification. For example: [{id: 'openPage1', name: 'Open Page 1'}, {id: 'openPage2', name: 'Open Page 2'}]

```
setActions(actions: INotificationActionPayload[]): void;
```

Parameters

Parameter Name	Required?	Description
title	Yes	The title of the notification.

INotificationCloseActionData

Functions

getNotificationId

Function to get notification id in action trigger event subscription response

```
getNotificationId(): string;
```

INotificationActionData

Functions

getNotificationId

Function to get notification id in action trigger event subscription response

```
getNotificationId(): string;
```

getActionName

Function to get action id name action trigger event subscription response.

```
getActionName(): string;
```

getActionId

Function to get action id action trigger event subscription response

```
getActionId(): string;
```

IMcaActionRequest

Functions

setAppClassification

Function to set appClassification.

```
setAppClassification: (appClassification: string) => void;
```

setEventId

Function to set event id.

```
setEventId: (eventId: string) => void;
```

IMcaNewCommEventActionRequest

NewCommEvent operation's request object.

Functions

setLookupObject

Function to set lookup object.

```
setLookupObject: (lookupObject: string) => void;
```

setInputData

Function to set inData object

```
setInputData: (inData: IMcaCloseCommInData) => void;
```

getInData

Function to get inData.

```
getInData: () => IMcaCloseCommInDataRequest;
```

IMcaStartCommEventActionRequest

StartCommEvent operation's request object.

Functions

setInputData

Function to set inData object

```
setInputData: (inData: IMcaCloseCommInData) => void;
```

getInData

Function to get inData.

```
getInData: () => IMcaCloseCommInDataRequest;
```

IMcaCloseComActionData

IMcaCloseCommEventActionRequest

CloseCommEvent operation's request object.

Functions

setReason

Function to set reason in request object.

```
setReason: (reason: string) => void;
```

setInputData

Function to set inData object

```
setInputData: (inData: IMcaCloseCommInData) => void;
```

getInData

Function to get inData.

```
getInData: () => IMcaCloseCommInDataRequest;
```

IMcaFeedLiveTranscriptActionRequest

This represents the FeedLiveTranscript operation's request object.

Functions

setMessage

Function to set the transcript message.

```
setMessage(message: string): void;
```

setMessageId

Function to set the message ID for a transcript message.

```
setMessageId(messageId: string): void;
```

setRole

Function to set the role for a transcript message.

```
setRole(role: string): void; // role can be 'AGENT' or 'END_USER'
```

Set the role as **AGENT** if the transcript message is from an agent.

Set the role as **END_USER** if the transcript message is from an end user.

setState

Function to set the state for a transcript message.

```
setState(state: string): void; // state can be 'STARTED', 'INPROGRESS' or 'COMPLETE'
```

Set the state as **STARTED** if the result of the transcription from the audio to text service isn't yet received.

Set the state as **INPROGRESS** if the result of the transcription from the audio to text service is partial or not final.

Set the state as **COMPLETE** if the result of the transcription from the audio to text service is complete or final.

setTime

Function to set the time for a transcript message.

```
setTime(time: number): void; // time as the epoch time. eg: Date.now()
```

IMcaFeedSystemMessageActionRequest

This represents the FeedSystemMessage operation's request object.

Functions

setMessage

Function to set the system message.

```
setMessage(message: string): void;
```

setMessageId

Function to set the message ID for a system message.

```
setMessageId(messageId: string): void;
```

setType

Function to set the type for the system message.

```
setType(type: string): void;
```

SetType message types

Type	Message Format
WorkJoin	<name> joined the call.
WorkLeave	<name> left the call.
WorkJoin_Monitor	The supervisor <name> is now observing the call.
WorkLeave_Monitor	The supervisor <name> has stopped observing the call.
WorkJoin_Conferee	The supervisor <name> has joined the call.
WorkLeave_Conferee	The supervisor <name> has left the call.
SYSTEM_MESSAGE	we can add custom messages here.

setIconName

Function to set the icon to display in the system message.

```
setIconName(iconName: string): void;
```

setUsername

Function to set the user name of the system message.

```
setUsername(username: string): void;
```

setUserId

Function to set the user ID of the system message.

```
setUserId(userId: string): void;
```

setTime

Function to set the time of the system message.

```
setTime(time: number): void;
```

IMcaNewComActionResponse

Response of newCommEvent operation.

Functions

getResponseData

Function to get response data in response.

```
getResponseData(): IMcaCloseComActionData;
```

IMcaStartComActionResponse

Response of startCommEvent operation.

Functions

getResponseData

Function to get response data in response.

```
getResponseData(): IMcaCloseComActionData;
```

IMcaCloseComActionResponse

Response of closeCommEvent operation.

Functions

getResponseData

Function to get response data in response.

```
getResponseData(): IMcaCloseComActionData;
```

IMcaNewComActionData

Data in StartCommEvent response.

Functions

getData

Data object in response.

```
getData(): IMcaStartComEventActionData;
```

getOutputData

Function to get entire output data in response

```
getOutputData(): IMcaStartCommEventOutDataResponse;
```

getOutData

Function to get outData in response.

```
setLookupObject: (value: string) => void;
```

getEngagementContext

Function to get engagementContext in response.

```
getEngagementContext(): IEngagementContext;
```

IMcaStartComActionData

Data in CloseCommEvent response.

Functions

getData

Data object in response.

```
getData(): IMcaStartComEventActionData;
```

getOutputData

Function to get entire output data in response

```
getOutputData(): IMcaStartCommEventOutDataResponse;
```

getOutData

Function to get outData in response.

```
setLookupObject: (value: string) => void;
```

IMcaEngagementData

Engagement data object in the return value of getEngagementData API of StartCommEvent response.

Functions

getData

Data object in response.

```
getData(): IMcaStartComEventActionData;
```

getOutputData

Function to get entire output data in response

```
getOutputData(): IMcaStartCommEventOutDataResponse;
```

getEngagementId

Function to get unique identifier of engagement object.

```
getEngagementId(): string;
```

setNotificationType

Function to get the setNotificationType value.

```
setNotificationType: (value: string) => void;
```

getOutData

Function to get outData in response.

```
setLookupObject: (value: string) => void;
```

IMcaOutData

Generic outData object in MCA operations response.

Object structure

Here's the structure of the of IMcaOutData:

```
interface IMcaOutData {
 SVCMCA_EMAIL: string;
 SVCMCA_COMMUNICATION_DIRECTION: string;
 SVCMCA_CONTACT_NAME: string;
 channel: string;
 channelType: string;
 SVCMCA_DISPLAY_NAME: string;
 notificationType: string;
 lookupObject: string;
 SVCMCA_OFFER_TIMEOUT_SEC: string;
 SVCMCA_CONTACT_FIRST_NAME: string;
 SVCMCA_SR_ID: string;
 appClassification: string;
 callStatus: string;
 channelId: string;
 eventId: string;
 SVCMCA_CONTACT_PRIMARY_PHONE: string;
 SVCMCA_INTERACTION_REF_OBJ_TYPE: string;
 SVCMCA_SR_NUM: string;
 SVCMCA_WRAPUP_TIMEOUT: string;
 SVCMCA_CONTACT_LAST_NAME: string;
 phoneLineId: string;
 SVCMCA_CONTACT_NUMBER: string;
 callDrection: string;
 SVCMCA_CONTACT_ID: string;
 SVCMCA_ANI: string;
 SVCMCA_CONTACT_PRIM_ORG_NAME: string;
 SVCMCA_INTERACTION_REF_OBJ_ID: string;
 SVCMCA_CONTACT_ORG_ID: string;
 SVCMCA_INTERACTION_ID: string;
 uefToken: string;
 }
```

IMcaStartCommEventOutData

OutData object in startCommEvent response. This object extends IMcaOutData.

Object structure

Here's the structure of the of IMcaStartCommEventOutData:

```
interface IMcaStartCommEventOutData extends IMcaOutData {
 SVCMCA_CHANNEL_ID: string;
 SVCMCA_UI_TYPE_CD: string;
 SVCMCA_SR_TITLE: string;
 SVCMCA_WRAPUP_ID: string;
 }
```

IMcaCloseCommEventOutData

OutData object in closeCommEvent response. This object extends IMcaOutData.

Object structure

Here's the structure of the of IMcaCloseCommEventOutData:

```
interface IMcaCloseCommEventOutData extends IMcaOutData {
 SVCMCA_CHANNEL_ID: string;
 SVCMCA_UI_TYPE_CD: string;
 SVCMCA_SR_TITLE: string;
 SVCMCA_WRAPUP_ID: string;
 WrapupStartTime: string;
 }
```

IMcaInData

Generic inData object in MCA operations.

Object structure

Here's the structure of the of IMcaInData:

```
interface IMcaInData {
 eventId: string;
 callStatus: string;
 callDirection: string;
 appClassification: string;
 notificationType: string;
 lookupObject: string;
 callSource: string;
 callDestination: string;
 channelType: string;
 comPanelMsg1: string;
 comPanelMsg2: string;
 comPanelQName1: string;
 comPanelQName2: string;
 comPanelQName3: string;
 averageWaitTime: string;
 numberOfCalls: string;
 agentMsg2ComPanel: string;
 phoneLineId: string;
 chatTestMode: string;
```

```
 SVCMCA_ANI?: string;
 SVCMCA_OFFER_TIMEOUT_SEC?: string;
 channel: string;
 SVCMCA_SR_NUM?: string;
 SVCMCA_COMMUNICATION_DIRECTION: string;
 SVCMCA_WRAPUP_TIMEOUT?: string;
 }
```

IMcaStartCommInData

Generic inData object in MCA startCommEvent operation.

Object structure

Here's the structure of the of IMcaStartCommInData:

```
interface IMcaStartCommInData extends IMcaInData {
 SVCMCA_EMAIL: string;
 SVCMCA_CONTACT_NAME: string;
 channel: string;
 SVCMCA_DISPLAY_NAME: string;
 SVCMCA_CONTACT_FIRST_NAME: string;
 SVCMCA_SR_ID: string;
 channelId: string;
 SVCMCA_CONTACT_PRIMARY_PHONE: string;
 SVCMCA_INTERACTION_REF_OBJ_TYPE: string;
 SVCMCA_CONTACT_LAST_NAME: string;
 SVCMCA_CONTACT_NUMBER: string;
 SVCMCA_CONTACT_ID: string;
 SVCMCA_CONTACT_PRIM_ORG_NAME: string;
 SVCMCA_INTERACTION_REF_OBJ_ID: string;
 SVCMCA_CONTACT_ORG_ID: string;
 SVCMCA_INTERACTION_ID: string;
 }
```

IMcaCloseCommInData

Generic inData object in MCA closeCommEvent operation.

Object structure

Here's the structure of the of IMcaCloseCommInData:

```
interface IMcaCloseCommInData extends IMcaInData {
 SVCMCA_EMAIL: string;
 SVCMCA_CONTACT_NAME: string;
 channel: string;
 SVCMCA_DISPLAY_NAME: string;
 SVCMCA_CONTACT_FIRST_NAME: string;
 SVCMCA_SR_ID: string;
 channelId: string;
```

```
SVCMCA_CONTACT_PRIMARY_PHONE: string;
SVCMCA_INTERACTION_REF_OBJ_TYPE: string;
SVCMCA_CONTACT_LAST_NAME: string;
SVCMCA_CONTACT_NUMBER: string;
SVCMCA_CONTACT_ID: string;
SVCMCA_CONTACT_PRIM_ORG_NAME: string;
SVCMCA_INTERACTION_REF_OBJ_ID: string;
SVCMCA_CONTACT_ORG_ID: string;
SVCMCA_INTERACTION_ID: string;
SVCMCA_CHANNEL_ID: string;
SVCMCA_UI_TYPE_CD: string;
SVCMCA_SR_TITLE: string;
SVCMCA_WRAPUP_ID: string;
}
```

IMcaInDataRequest

Generic inData object in MCA operations.

Functions

setCallDestination

Function to set the CallDestination value.

```
setCallDestination: (value: string) => void;
```

setCallDirection

Function to set the CallDirection value.

```
setCallDirection: (value: string) => void;
```

setAppClassification

Function to set the AppClassification value.

```
setAppClassification: (value: string) => void;
```

setNotificationType

Function to get the setNotificationType value.

```
setNotificationType: (value: string) => void;
```

setLookupObject

Function to set the LookupObject value.

```
setLookupObject: (value: string) => void;
```

setCallSource

Function to set the CallSource value.

```
setCallSource: (value: string) => void;
```

setCallStatus

Function to set the CallStatus value.

```
setCallStatus: (value: string) => void;
```

setChannelType

Function to set the ChannelType value.

```
setChannelType: (value: string) => void;
```

setComPanelMsg1

Function to set the ComPanelMsg1 value.

```
setComPanelMsg1: (value: string) => void;
```

setComPanelMsg2

Function to set the ComPanelMsg2 value.

```
setComPanelMsg2: (value: string) => void;
```

setComPanelQName1

Function to set the ComPanelQName1 value.

```
setComPanelQName1: (value: string) => void;
```

setComPanelQName2

Function to set the ComPanelQName2 value.

```
setComPanelQName2: (value: string) => void;
```

setComPanelQName3

Function to set the ComPanelQName3 value.

```
setComPanelQName3: (value: string) => void;
```

setAverageWaitTime

Function to set the AverageWaitTime value.

```
setAverageWaitTime: (value: string) => void;
```

setNumberOfCalls

Function to set the NumberOfCalls value.

```
setNumberOfCalls: (value: string) => void;
```

setAgentMsg2ComPanel

Function to set the AgentMsg2ComPanel value.

```
setAgentMsg2ComPanel: (value: string) => void;
```

setPhoneLineId

Function to set the PhoneLineId value.

```
setPhoneLineId: (value: string) => void;
```

setChatTestMode

Function to set the ChatTestMode value.

```
setChatTestMode: (value: string) => void;
```

setSVCMCA_ANI

Function to set the SVCMCA_ANI value.

```
setSVCMCA_ANI: (value: string) => void;
```

setSVCMCA_WRAPUP_TIMEOUT

Function to set the SVCMCA_WRAPUP_TIMEOUT value.

```
setSVCMCA_WRAPUP_TIMEOUT: (value: string) => void;
```

setSVCMCA_OFFER_TIMEOUT_SEC

Function to set the SVCMCA_OFFER_TIMEOUT_SEC value.

```
setSVCMCA_OFFER_TIMEOUT_SEC: (value: string) => void;
```

setSVCMCA_COMMUNICATION_DIRECTION

Function to set the SVCMCA_COMMUNICATION_DIRECTION value.

```
setSVCMCA_COMMUNICATION_DIRECTION: (value: string) => void;
```

setSVCMCA_SR_NUM

Function to set the SVCMCA_SR_NUM value.

```
setSVCMCA_SR_NUM: (value: string) => void;
```

setChannel

Function to set the Channel.

```
setChannel: (value: string) => void;
```

setInDataValueByAttribute

Function to setInDataValueByAttribute value.

```
setInDataValueByAttribute: (attribute: string, value: any) => void;
```

IMcaStartCommInDataRequest

Generic inData object in MCA StartCommEvent InData request object.

Functions

getSVCMCA_EMAIL

Function to get SVCMCA_EMAIL.

```
setSVCMCA_EMAIL: (value: string) => void;
```

getSVCMCA_CONTACT_NAME

Function to get SVCMCA_CONTACT_NAME.

```
setSVCMCA_CONTACT_NAME: (value: string) => void;
```

setChannel

Function to set the Channel.

```
setChannel: (value: string) => void;
```

setSVCMCA_DISPLAY_NAME

Function to set the SVCMCA_DISPLAY_NAME value.

```
setSVCMCA_DISPLAY_NAME: (value: string) => void;
```

getSVCMCA_CONTACT_FIRST_NAME

Function to get the SVCMCA_CONTACT_FIRST_NAME value.

```
setSVCMCA_CONTACT_FIRST_NAME: (value: string) => void;
```

setSVCMCA_SR_ID

Function to set the SVCMCA_SR_ID.

```
setSVCMCA_SR_ID: (value: string) => void;
```

setChannelId

Function to set the ChannelId.

```
setChannelId: (value: string) => void;
```

getSVCMCA_CONTACT_PRIMARY_PHONE

Function to get the SVCMCA_CONTACT_PRIMARY_PHONE.

```
setSVCMCA_CONTACT_PRIMARY_PHONE: (value: string) => void;
```

setSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to set the SVCMCA_INTERACTION_REF_OBJ_TYPE value.

```
setSVCMCA_INTERACTION_REF_OBJ_TYPE: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_LAST_NAME

Function to get the SVCMCA_CONTACT_LAST_NAME value.

```
setSVCMCA_CONTACT_LAST_NAME: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_NUMBER

Function to get the SVCMCA_CONTACT_NUMBER value.

```
setSVCMCA_CONTACT_NUMBER: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_ID

Function to get the SVCMCA_CONTACT_ID value.

```
setSVCMCA_CONTACT_ID: (value: string) =&gt; void;
```

setSVCMCA_CONTACT_PRIM_ORG_NAME

Function to set the SVCMCA_CONTACT_PRIM_ORG_NAME value.

```
getChannelType: () => string;
```

setSVCMCA_INTERACTION_REF_OBJ_ID

Function to get the setSVCMCA_INTERACTION_REF_OBJ_ID value.

```
setSVCMCA_INTERACTION_REF_OBJ_ID: (value: string) =&gt; void;
```

setSVCMCA_CONTACT_ORG_ID

Function to set the SVCMCA_CONTACT_ORG_ID value.

```
setSVCMCA_CONTACT_ORG_ID: (value: string) => void;
```

setSVCMCA_INTERACTION_ID

Function to set the SVCMCA_INTERACTION_ID value.

```
setSVCMCA_INTERACTION_ID: (value: string) => void;
```

setCallDestination

Function to set the CallDestination value.

```
setCallDestination: (value: string) => void;
```

setCallDirection

Function to set the CallDirection value.

```
setCallDirection: (value: string) => void;
```

setAppClassification

Function to set the AppClassification value.

```
setAppClassification: (value: string) => void;
```

setNotificationType

Function to get the setNotificationType value.

```
setNotificationType: (value: string) => void;
```

setLookupObject

Function to set the LookupObject value.

```
setLookupObject: (value: string) => void;
```

setCallSource

Function to set the CallSource value.

```
setCallSource: (value: string) => void;
```

setCallStatus

Function to set the CallStatus value.

```
setCallStatus: (value: string) => void;
```

setChannelType

Function to set the ChannelType value.

```
setChannelType: (value: string) => void;
```

setComPanelMsg1

Function to set the ComPanelMsg1 value.

```
setComPanelMsg1: (value: string) => void;
```

setComPanelMsg2

Function to set the ComPanelMsg2 value.

```
setComPanelMsg2: (value: string) => void;
```

setComPanelQName1

Function to set the ComPanelQName1 value.

```
setComPanelQName1: (value: string) => void;
```

setComPanelQName2

Function to set the ComPanelQName2 value.

```
setComPanelQName2: (value: string) => void;
```

setComPanelQName3

Function to set the ComPanelQName3 value.

```
setComPanelQName3: (value: string) => void;
```

setAverageWaitTime

Function to set the AverageWaitTime value.

```
setAverageWaitTime: (value: string) => void;
```

setNumberOfCalls

Function to set the NumberOfCalls value.

```
setNumberOfCalls: (value: string) => void;
```

setAgentMsg2ComPanel

Function to set the AgentMsg2ComPanel value.

```
setAgentMsg2ComPanel: (value: string) => void;
```

setPhoneLineId

Function to set the PhoneLineId value.

```
setPhoneLineId: (value: string) => void;
```

setChatTestMode

Function to set the ChatTestMode value.

```
setChatTestMode: (value: string) => void;
```

setSVCMCA_ANI

Function to set the SVCMCA_ANI value.

```
setSVCMCA_ANI: (value: string) => void;
```

setSVCMCA_WRAPUP_TIMEOUT

Function to set the SVCMCA_WRAPUP_TIMEOUT value.

```
setSVCMCA_WRAPUP_TIMEOUT: (value: string) => void;
```

setSVCMCA_OFFER_TIMEOUT_SEC

Function to set the SVCMCA_OFFER_TIMEOUT_SEC value.

```
setSVCMCA_OFFER_TIMEOUT_SEC: (value: string) => void;
```

setSVCMCA_COMMUNICATION_DIRECTION

Function to set the SVCMCA_COMMUNICATION_DIRECTION value.

```
setSVCMCA_COMMUNICATION_DIRECTION: (value: string) => void;
```

setSVCMCA_SR_NUM

Function to set the SVCMCA_SR_NUM value.

```
setSVCMCA_SR_NUM: (value: string) => void;
```

setInDataValueByAttribute

Function to setInDataValueByAttribute value.

```
setInDataValueByAttribute: (attribute: string, value: any) => void;
```

IMcaCloseCommInDataRequest

Generic inData object in MCA CloseCommEvent InData request object.

Functions

getSVCMCA_EMAIL

Function to get SVCMCA_EMAIL.

```
setSVCMCA_EMAIL: (value: string) => void;
```

getSVCMCA_CONTACT_NAME

Function to get SVCMCA_CONTACT_NAME.

```
setSVCMCA_CONTACT_NAME: (value: string) => void;
```

setChannel

Function to set the Channel.

```
setChannel: (value: string) => void;
```

setSVCMCA_DISPLAY_NAME

Function to set the SVCMCA_DISPLAY_NAME value.

```
setSVCMCA_DISPLAY_NAME: (value: string) => void;
```

getSVCMCA_CONTACT_FIRST_NAME

Function to get the SVCMCA_CONTACT_FIRST_NAME value.

```
setSVCMCA_CONTACT_FIRST_NAME: (value: string) => void;
```

setSVCMCA_SR_ID

Function to set the SVCMCA_SR_ID.

```
setSVCMCA_SR_ID: (value: string) => void;
```

setChannelId

Function to set the ChannelId.

```
setChannelId: (value: string) => void;
```

getSVCMCA_CONTACT_PRIMARY_PHONE

Function to get the SVCMCA_CONTACT_PRIMARY_PHONE.

```
setSVCMCA_CONTACT_PRIMARY_PHONE: (value: string) => void;
```

setSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to set the SVCMCA_INTERACTION_REF_OBJ_TYPE value.

```
setSVCMCA_INTERACTION_REF_OBJ_TYPE: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_LAST_NAME

Function to get the SVCMCA_CONTACT_LAST_NAME value.

```
setSVCMCA_CONTACT_LAST_NAME: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_NUMBER

Function to get the SVCMCA_CONTACT_NUMBER value.

```
setSVCMCA_CONTACT_NUMBER: (value: string) =&gt; void;
```

getSVCMCA_CONTACT_ID

Function to get the SVCMCA_CONTACT_ID value.

```
setSVCMCA_CONTACT_ID: (value: string) =&gt; void;
```

setSVCMCA_CONTACT_PRIM_ORG_NAME

Function to set the SVCMCA_CONTACT_PRIM_ORG_NAME value.

```
getChannelType: () => string;
```

setSVCMCA_INTERACTION_REF_OBJ_ID

Function to get the setSVCMCA_INTERACTION_REF_OBJ_ID value.

```
setSVCMCA_INTERACTION_REF_OBJ_ID: (value: string) =&gt; void;
```

setSVCMCA_CONTACT_ORG_ID

Function to set the SVCMCA_CONTACT_ORG_ID value.

```
setSVCMCA_CONTACT_ORG_ID: (value: string) => void;
```

setSVCMCA_INTERACTION_ID

Function to set the SVCMCA_INTERACTION_ID value.

```
setSVCMCA_INTERACTION_ID: (value: string) => void;
```

setSVCMCA_UI_TYPE_CD

Function to set the SVCMCA_UI_TYPE_CD value.

```
setSVCMCA_UI_TYPE_CD: (value: string) => void;
```

setSVCMCA_CHANNEL_ID

Function to set the SVCMCA_CHANNEL_ID value.

```
etSVCMCA_CHANNEL_ID: (value: string) => void;
```

setSVCMCA_SR_TITLE

Function to set the SVCMCA_SR_TITLE value.

```
setSVCMCA_SR_TITLE: (value: string) => void;
```

setSVCMCA_WRAPUP_ID

Function to set the SVCMCA_WRAPUP_ID value.

```
setSVCMCA_WRAPUP_ID: (value: string) => void;
```

setCallDestination

Function to set the CallDestination value.

```
setCallDestination: (value: string) => void;
```

setCallDirection

Function to set the CallDirection value.

```
setCallDirection: (value: string) => void;
```

setAppClassification

Function to set the AppClassification value.

```
setAppClassification: (value: string) => void;
```

setNotificationType

Function to get the setNotificationType value.

```
setNotificationType: (value: string) => void;
```

setLookupObject

Function to set the LookupObject value.

```
setLookupObject: (value: string) => void;
```

setCallSource

Function to set the CallSource value.

```
setCallSource: (value: string) => void;
```

setCallStatus

Function to set the CallStatus value.

```
setCallStatus: (value: string) => void;
```

setChannelType

Function to set the ChannelType value.

```
setChannelType: (value: string) => void;
```

setComPanelMsg1

Function to set the ComPanelMsg1 value.

```
setComPanelMsg1: (value: string) => void;
```

setComPanelMsg2

Function to set the ComPanelMsg2 value.

```
setComPanelMsg2: (value: string) => void;
```

setComPanelQName1

Function to set the ComPanelQName1 value.

```
setComPanelQName1: (value: string) => void;
```

setComPanelQName2

Function to set the ComPanelQName2 value.

```
setComPanelQName2: (value: string) => void;
```

setComPanelQName3

Function to set the ComPanelQName3 value.

```
setComPanelQName3: (value: string) => void;
```

setAverageWaitTime

Function to set the AverageWaitTime value.

```
setAverageWaitTime: (value: string) => void;
```

setNumberOfCalls

Function to set the NumberOfCalls value.

```
setNumberOfCalls: (value: string) => void;
```

setAgentMsg2ComPanel

Function to set the AgentMsg2ComPanel value.

```
setAgentMsg2ComPanel: (value: string) => void;
```

setPhoneLineId

Function to set the PhoneLineId value.

```
setPhoneLineId: (value: string) => void;
```

setChatTestMode

Function to set the ChatTestMode value.

```
setChatTestMode: (value: string) => void;
```

setSVCMCA_ANI

Function to set the SVCMCA_ANI value.

```
setSVCMCA_ANI: (value: string) => void;
```

setSVCMCA_WRAPUP_TIMEOUT

Function to set the SVCMCA_WRAPUP_TIMEOUT value.

```
setSVCMCA_WRAPUP_TIMEOUT: (value: string) => void;
```

setSVCMCA_OFFER_TIMEOUT_SEC

Function to set the SVCMCA_OFFER_TIMEOUT_SEC value.

```
setSVCMCA_OFFER_TIMEOUT_SEC: (value: string) => void;
```

setSVCMCA_COMMUNICATION_DIRECTION

Function to set the SVCMCA_COMMUNICATION_DIRECTION value.

```
setSVCMCA_COMMUNICATION_DIRECTION: (value: string) => void;
```

setSVCMCA_SR_NUM

Function to set the SVCMCA_SR_NUM value.

```
setSVCMCA_SR_NUM: (value: string) => void;
```

setInDataValueByAttribute

Function to setInDataValueByAttribute value.

```
setInDataValueByAttribute: (attribute: string, value: any) => void;
```

IMcaComEventActionData

Generic response object in MCA operations.

Object structure

Here's the structure of the of IMcaComEventActionData:

```
interface IMcaComEventActionData {
 channelId: string;
 channelType: string;
 engagementId: string;
 eventId: string;
 eventSource: string;
 interactionId: string;
 method: string;
 result: string;
 toolbarName: string;
 uuid: string;
 channel: McaChannels;
 }
```

IMcaNewComEventActionData

Mca NewCommEvent operation response data, which also extends IMcaComEventActionData.

```
interface IMcaNewComEventActionData extends IMcaComEventActionData {
 outData: IMcaOutData;
 }
```

Attributes

outdata

OutData attribute in closeCommEvent operation Response.

```
outData: IMcaOutData;
```

IMcaStartComEventActionData

Mca StartCommEvent operation response data, which also extends IMcaComEventActionData.

```
interface IMcaStartComEventActionData extends IMcaComEventActionData {
 outData: IMcaStartCommEventOutData;
 screenPopMode: string;
 wrapupId: string;
 }
```

Attributes

outdata

OutData object in StartCommEvent response.

```
outData: IMcaStartCommEventOutData;
```

screenPopMode

ScreenPop attribute in startCommEvent response.

```
screenPopMode: string;
```

wrapupId

WrapupId attribute in startCommEvent operation Response.

```
wrapupId: string;
```

IMcaCloseComEventActionData

Mca CloseCommEvent operation response data, which also extends IMcaCloseComEventActionData.

```
interface IMcaCloseComEventActionData extends IMcaComEventActionData {
 outData: IMcaCloseCommEventOutData;
 wrapupId: string;
 }
```

Attributes

outdata

OutData attribute in closeCommEvent operation Response.

```
outData: IMcaCloseCommEventOutData;
```

wrapupId

WrapupId attribute in closeCommEvent operation Response.

```
wrapupId: string;
```

IMcaOutDataResponse

Generic outData response object for MCA operation response.

Functions

getSVCMCA_COMMUNICATION_DIRECTION

Function to get SVCMCA_COMMUNICATION_DIRECTION.

```
getSVCMCA_COMMUNICATION_DIRECTION: () => string;
```

getSVCMCA_CONTACT_NAME

Function to get SVCMCA_CONTACT_NAME.

```
getSVCMCA_CONTACT_NAME: () => string;
```

getSVCMCA_EMAIL

Function to get SVCMCA_EMAIL.

```
getSVCMCA_EMAIL: () => string;
```

getChannel

Function to get the Channel.

```
getChannel: () => string;
```

getChannelType

Function to get the Channel Type.

```
getChannelType: () => string;
```

getSVCMCA_DISPLAY_NAME

Function to get getSVCMCA_DISPLAY_NAME.

```
getSVCMCA_DISPLAY_NAME: () => string;
```

getNotificationType

Function to get the Notification Type.

```
getNotificationType: () => string
```

getLookupObject

Function to get the Lookup Object.

```
getLookupObject: () => string;
```

getSVCMCA_OFFER_TIMEOUT_SEC

Function to set the getSVCMCA_OFFER_TIMEOUT_SEC value.

```
getSVCMCA_OFFER_TIMEOUT_SEC: () => string;
```

getSVCMCA_CONTACT_FIRST_NAME

Function to get the SVCMCA_CONTACT_FIRST_NAME value.

```
getSVCMCA_CONTACT_FIRST_NAME: () => string;
```

getSVCMCA_SR_ID

Function to get the SVCMCA_SR_ID value.

```
getSVCMCA_SR_ID: () => string;
```

getAppClassification

Function to get the AppClassification.

```
getAppClassification: () => string;
```

getCallStatus

Function to get the CallStatus value.

```
getCallStatus: () => string;
```

getChannelId

Function to get the channel Id.

```
getChannelId: () => string;
```

getEventId

Function to get the Event Id.

```
getEventId: () => string;
```

getSVCMCA_CONTACT_PRIMARY_PHONE

Function to get the SVCMCA_CONTACT_PRIMARY_PHONE.

```
getSVCMCA_CONTACT_PRIMARY_PHONE: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to get SVCMCA_INTERACTION_REF_OBJ_TYPE value.

```
getSVCMCA_INTERACTION_REF_OBJ_TYPE: () => string;
```

getSVCMCA_SR_NUM

Function to get the SVCMCA_SR_NUM value.

```
getEventId: () => string;
```

getSVCMCA_WRAPUP_TIMEOUT

Function to get the getSVCMCA_WRAPUP_TIMEOUT value.

```
getSVCMCA_WRAPUP_TIMEOUT: () => string;
```

getSVCMCA_CONTACT_LAST_NAME

Function to get the SVCMCA_CONTACT_LAST_NAME value.

```
getSVCMCA_CONTACT_LAST_NAME: () => string;
```

getPhoneLineId

Function to get the PhoneLineId value.

```
getPhoneLineId: () => string;
```

getSVCMCA_CONTACT_NUMBER

Function to get the SVCMCA_CONTACT_NUMBER value.

```
getSVCMCA_CONTACT_NUMBER: () => string;
```

getCallDirection

Function to get the CallDirection value.

```
getCallDirection: () => string;
```

getSVCMCA_CONTACT_ID

Function to get the SVCMCA_CONTACT_ID value.

```
getSVCMCA_CONTACT_ID: () => string;
```

getSVCMCA_ANI

Function to get the SVCMCA_ANI value.

```
getSVCMCA_ANI: () => string;
```

getSVCMCA_CONTACT_PRIM_ORG_NAME

Function to get the SVCMCA_CONTACT_PRIM_ORG_NAME value.

```
getSVCMCA_CONTACT_PRIM_ORG_NAME: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_ID

Function to get the SVCMCA_INTERACTION_REF_OBJ_ID value.

```
getSVCMCA_INTERACTION_REF_OBJ_ID: () => string;
```

getSVCMCA_CONTACT_ORG_ID

Function to get the SVCMCA_CONTACT_ORG_ID value.

```
getSVCMCA_CONTACT_ORG_ID: () => string;
```

getSVCMCA_INTERACTION_ID

Function to get the SVCMCA_INTERACTION_ID value.

```
getSVCMCA_INTERACTION_ID: () => string;
```

getValueByAttribute

Function to get the ValueByAttribute value.

```
getValueByAttribute: (attribute: string) => string;
```

IMcaCloseCommEventOutDataResponse

Generic outData response object for MCA CloseCommEvent operation response.

Functions

getSVCMCA_COMMUNICATION_DIRECTION

Function to get SVCMCA_COMMUNICATION_DIRECTION.

```
getSVCMCA_COMMUNICATION_DIRECTION: () => string;
```

getSVCMCA_CONTACT_NAME

Function to get SVCMCA_CONTACT_NAME.

```
getSVCMCA_CONTACT_NAME: () => string;
```

getSVCMCA_EMAIL

Function to get SVCMCA_EMAIL.

```
getSVCMCA_EMAIL: () => string;
```

getChannel

Function to get the Channel.

```
getChannel: () => string;
```

getChannelType

Function to get the Channel Type.

```
getChannelType: () => string;
```

getSVCMCA_DISPLAY_NAME

Function to get getSVCMCA_DISPLAY_NAME.

```
getSVCMCA_DISPLAY_NAME: () => string;
```

getNotificationType

Function to get the Notification Type.

```
getNotificationType: () => string
```

getLookupObject

Function to get the Lookup Object.

```
getLookupObject: () => string;
```

getSVCMCA_OFFER_TIMEOUT_SEC

Function to set the getSVCMCA_OFFER_TIMEOUT_SEC value.

```
getSVCMCA_OFFER_TIMEOUT_SEC: () => string;
```

getSVCMCA_CONTACT_FIRST_NAME

Function to get the SVCMCA_CONTACT_FIRST_NAME value.

```
getSVCMCA_CONTACT_FIRST_NAME: () => string;
```

getSVCMCA_SR_ID

Function to get the SVCMCA_SR_ID value.

```
getSVCMCA_SR_ID: () => string;
```

getAppClassification

Function to get the AppClassification.

```
getAppClassification: () => string;
```

getCallStatus

Function to get the CallStatus value.

```
getCallStatus: () => string;
```

getChannelId

Function to get the channel Id.

```
getChannelId: () => string;
```

getEventId

Function to get the Event Id.

```
getEventId: () => string;
```

getSVCMCA_CONTACT_PRIMARY_PHONE

Function to get the SVCMCA_CONTACT_PRIMARY_PHONE.

```
getSVCMCA_CONTACT_PRIMARY_PHONE: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to get SVCMCA_INTERACTION_REF_OBJ_TYPE value.

```
getSVCMCA_INTERACTION_REF_OBJ_TYPE: () => string;
```

getSVCMCA_SR_NUM

Function to get the SVCMCA_SR_NUM value.

```
getEventId: () => string;
```

getSVCMCA_WRAPUP_TIMEOUT

Function to get the getSVCMCA_WRAPUP_TIMEOUT value.

```
getSVCMCA_WRAPUP_TIMEOUT: () => string;
```

getSVCMCA_CONTACT_LAST_NAME

Function to get the SVCMCA_CONTACT_LAST_NAME value.

```
getSVCMCA_CONTACT_LAST_NAME: () => string;
```

getPhoneLineId

Function to get the PhoneLineId value.

```
getPhoneLineId: () => string;
```

getSVCMCA_CONTACT_NUMBER

Function to get the SVCMCA_CONTACT_NUMBER value.

```
getSVCMCA_CONTACT_NUMBER: () => string;
```

getCallDirection

Function to get the CallDirection value.

```
getCallDirection: () => string;
```

getSVCMCA_CONTACT_ID

Function to get the SVCMCA_CONTACT_ID value.

```
getSVCMCA_CONTACT_ID: () => string;
```

getSVCMCA_ANI

Function to get the SVCMCA_ANI value.

```
getSVCMCA_ANI: () => string;
```

getSVCMCA_CONTACT_PRIM_ORG_NAME

Function to get the SVCMCA_CONTACT_PRIM_ORG_NAME value.

```
getSVCMCA_CONTACT_PRIM_ORG_NAME: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_ID

Function to get the SVCMCA_INTERACTION_REF_OBJ_ID value.

```
getSVCMCA_INTERACTION_REF_OBJ_ID: () => string;
```

getSVCMCA_CONTACT_ORG_ID

Function to get the SVCMCA_CONTACT_ORG_ID value.

```
getSVCMCA_CONTACT_ORG_ID: () => string;
```

getSVCMCA_INTERACTION_ID

Function to get the SVCMCA_INTERACTION_ID value.

```
getSVCMCA_INTERACTION_ID: () => string;
```

getValueByAttribute

Function to get the ValueByAttribute value.

```
getValueByAttribute: (attribute: string) => string;
```

getSVCMCA_CHANNEL_ID

Function to get the SVCMCA_CHANNEL_ID value.

```
getSVCMCA_CHANNEL_ID: () => string;
```

getSVCMCA_UI_TYPE_CD

Function to get the SVCMCA_UI_TYPE_CD value.

```
getSVCMCA_UI_TYPE_CD: () => string;
```

getSVCMCA_SR_TITLE

Function to get the SVCMCA_SR_TITLE value.

```
getSVCMCA_SR_TITLE: () => string;
```

getSVCMCA_WRAPUP_ID

Function to get the SVCMCA_WRAPUP_ID value.

```
getSVCMCA_WRAPUP_ID: () => string;
```

getWrapupStartTime

Function to get the getWrapupStartTime value.

```
getWrapupStartTime: () => string;
```

IMcaStartCommEventOutDataResponse

Generic outData response object for MCA StartCommEvent operation response

Functions

getSVCMCA_COMMUNICATION_DIRECTION

Function to get SVCMCA_COMMUNICATION_DIRECTION.

```
getSVCMCA_COMMUNICATION_DIRECTION: () => string;
```

getSVCMCA_CONTACT_NAME

Function to get SVCMCA_CONTACT_NAME.

```
getSVCMCA_CONTACT_NAME: () => string;
```

getSVCMCA_EMAIL

Function to get SVCMCA_EMAIL.

```
getSVCMCA_EMAIL: () => string;
```

getChannel

Function to get the Channel.

```
getChannel: () => string;
```

getChannelType

Function to get the Channel Type.

```
getChannelType: () => string;
```

getSVCMCA_DISPLAY_NAME

Function to get getSVCMCA_DISPLAY_NAME.

```
getSVCMCA_DISPLAY_NAME: () => string;
```

getNotificationType

Function to get the Notification Type.

```
getNotificationType: () => string
```

getLookupObject

Function to get the Lookup Object.

```
getLookupObject: () => string;
```

getSVCMCA_OFFER_TIMEOUT_SEC

Function to set the getSVCMCA_OFFER_TIMEOUT_SEC value.

```
getSVCMCA_OFFER_TIMEOUT_SEC: () => string;
```

getSVCMCA_CONTACT_FIRST_NAME

Function to get the SVCMCA_CONTACT_FIRST_NAME value.

```
getSVCMCA_CONTACT_FIRST_NAME: () => string;
```

getSVCMCA_SR_ID

Function to get the SVCMCA_SR_ID value.

```
getSVCMCA_SR_ID: () => string;
```

getAppClassification

Function to get the AppClassification.

```
getAppClassification: () => string;
```

getCallStatus

Function to get the CallStatus value.

```
getCallStatus: () => string;
```

getChannelId

Function to get the channel Id.

```
getChannelId: () => string;
```

getEventId

Function to get the Event Id.

```
getEventId: () => string;
```

getSVCMCA_CONTACT_PRIMARY_PHONE

Function to get the SVCMCA_CONTACT_PRIMARY_PHONE.

```
getSVCMCA_CONTACT_PRIMARY_PHONE: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to get SVCMCA_INTERACTION_REF_OBJ_TYPE value.

```
getSVCMCA_INTERACTION_REF_OBJ_TYPE: () => string;
```

getSVCMCA_SR_NUM

Function to get the SVCMCA_SR_NUM value.

```
getEventId: () => string;
```

getSVCMCA_WRAPUP_TIMEOUT

Function to get the getSVCMCA_WRAPUP_TIMEOUT value.

```
getSVCMCA_WRAPUP_TIMEOUT: () => string;
```

getSVCMCA_CONTACT_LAST_NAME

Function to get the SVCMCA_CONTACT_LAST_NAME value.

```
getSVCMCA_CONTACT_LAST_NAME: () => string;
```

getPhoneLineId

Function to get the PhoneLineId value.

```
getPhoneLineId: () => string;
```

getSVCMCA_CONTACT_NUMBER

Function to get the SVCMCA_CONTACT_NUMBER value.

```
getSVCMCA_CONTACT_NUMBER: () => string;
```

getCallDirection

Function to get the CallDirection value.

```
getCallDirection: () => string;
```

getSVCMCA_CONTACT_ID

Function to get the SVCMCA_CONTACT_ID value.

```
getSVCMCA_CONTACT_ID: () => string;
```

getSVCMCA_ANI

Function to get the SVCMCA_ANI value.

```
getSVCMCA_ANI: () => string;
```

getSVCMCA_CONTACT_PRIM_ORG_NAME

Function to get the SVCMCA_CONTACT_PRIM_ORG_NAME value.

```
getSVCMCA_CONTACT_PRIM_ORG_NAME: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_ID

Function to get the SVCMCA_INTERACTION_REF_OBJ_ID value.

```
getSVCMCA_INTERACTION_REF_OBJ_ID: () => string;
```

getSVCMCA_CONTACT_ORG_ID

Function to get the SVCMCA_CONTACT_ORG_ID value.

```
getSVCMCA_CONTACT_ORG_ID: () => string;
```

getSVCMCA_INTERACTION_ID

Function to get the SVCMCA_INTERACTION_ID value.

```
getSVCMCA_INTERACTION_ID: () => string;
```

getValueByAttribute

Function to get the ValueByAttribute value.

```
getValueByAttribute: (attribute: string) => string;
```

getSVCMCA_CHANNEL_ID

Function to get the SVCMCA_CHANNEL_ID value.

```
getSVCMCA_CHANNEL_ID: () => string;
```

getSVCMCA_UI_TYPE_CD

Function to get the SVCMCA_UI_TYPE_CD value.

```
getSVCMCA_UI_TYPE_CD: () => string;
```

getSVCMCA_SR_TITLE

Function to get the SVCMCA_SR_TITLE value.

```
getSVCMCA_SR_TITLE: () => string;
```

getSVCMCA_WRAPUP_ID

Function to get the SVCMCA_WRAPUP_ID value.

```
getSVCMCA_WRAPUP_ID: () => string;
```

IMcaGetConfigurationActionRequest

This is the request object for MCA GetConfiguration operation

Functions

setConfigType

Function to set the configuration type.

```
setConfigType: (configType: IMCAGetConfigurationConfigTypes) => void;
```

IMcaAgentStateEventActionRequest

This is the request object for MCA AgentStateEvent operation.

Functions

setEventId

Function to set the EventId.

```
setEventId: (eventId: string) => void;
```

setIsAvailable

Function to set IsAvailable.

```
setIsAvailable: (isAvailable: boolean) => void;
```

setIsLoggedIn

Function to set setIsLoggedIn.

```
setIsLoggedIn: (isLoggedIn: boolean) => void;
```

setState

Function to set the State.

```
setState: (stateCd: string) => void;
```

setStateDisplayString

Function to set setStateDisplayString.

```
setStateDisplayString: (stateDisplayString: string) => void;
```

setReason

Function to set the reason.

```
setReason: (reasonCd: string) => void;
```

setReasonDisplayString

Function to set setReasonDisplayString.

```
setReasonDisplayString: (reasonDisplayString: string) => void;
```

setInData

Function to set InData.

```
setInData: (inData: Record<string, any>) => void;
```

IMcaDisableFeatureActionRequest

This is the request object for MCA DisableFeature operation.

Functions

setFeatures

Function to set features as an array of string.

```
setFeatures: (features: string[]) => void;
```

IMcaReadyForOperationActionRequest

This is the request object for MCA ReadyForOperation operation.

Functions

setReadiness

Function to set the readiness flag.

```
setReadiness: (readiness: boolean) => void;
```

IMcaGetConfigurationActionResponse

This is the response object for Get Configuration Response.

Functions

getResponseData

Function to get response data.

```
getResponseData(): IMcaGetConfigurationActionResponseData;
```

IMcaDisableFeatureActionResponse

This is the response object for Disable Feature Action Response.

Functions

IMcaDisableFeatureActionResponseData

Function to get response data.

```
getResponseData(): IMcaDisableFeatureActionResponseData;
```

IMcaReadyForOperationActionResponse

This is the response object for ready for operation response.

Functions

IMcaReadyForOperationActionResponseData

Function to get response data.

```
getResponseData(): IMcaReadyForOperationActionResponseData;
```

IMcaAgentStateEventActionResponse

This is the response object for AgentStateEvent response.

Functions

IMcaAgentStateEventActionResponseData

Function to get response data.

```
getResponseData(): IMcaAgentStateEventActionResponseData;
```

IMcaOutboundCommErrorActionRequest

This is the request object for Outbound Comm Error Action Request.

Functions

setCommUuid

Function to set the setCommUuid.

```
setCommUuid(commUuid: string): void;
```

IMcaOutBoundCommErrorActionResponse

This is the request object for Outbound Comm Error Action Request.

Functions

setCommUuid

Function to set the setCommUuid.

```
setCommUuid(commUuid: string): void;
```

setErrorCode

Function to set the setErrorCode.

```
setErrorCode(errorCode: string): void;
```

setErrorMsg

Function to set the setErrorMsg.

```
setErrorMsg(errorMsg: string): void;
```

IMcaGetConfigurationActionResponseData

This is the response object of Outbound Comm Error Operation.

Function

getResponseData

Function to get response data.

```
getResponseData(): IMcaOutBoundCommErrorActionResponseData;
```

IMcaOutBoundCommErrorActionResponseData

Response data object for GetConfiguration operation.

Function

configuration

Function to get configuration.

```
getConfiguration(): IMCAGetConfiguration;
```

isSuccess

Function to get success status.

```
isSuccess(): boolean;
```

getError

Function to get any error

```
getError(): any;
```

IMcaAgentStateEventActionResponseData

Response data object of AgentStateEvent operation.

Function

getData

Function to get data.

```
getData(): IMCAOutBoundCommErrorResponsePayload;
```

isSuccess

Function to get success status.

```
isSuccess(): boolean;
```

getError

Function to get error

```
getError(): string;
```

IMcaDisableFeatureActionResponseData

Response data object of Disable Feature operation.

Function

getData

Function to get data.

```
getData(): any;
```

IMcaReadyForOperationActionResponseData

Response data object of Ready for Operation operation.

Function

getData

Function to get data.

```
getData(): any;
```

IMCAGetConfigurationResponsePayload

Response data payload object of GetConfiguration operation.

Attributes

outData

Attribute which contains the outData object.

```
outData: Record<string, any>;
```

method

Attribute which contains method.

```
method: any;
```

configuration

Attribute which contains the configuration object.

```
configuration: IMCAConfiguration;
```

result

Attribute which contains the result.

```
result: string;
```

toolbarName

Attribute which contains the toolbar Name.

```
toolbarName: string;
```

uuid

Attribute which contains information on uuid.

```
uuid: string;
```

error

Attribute which contains the error, if any.

```
error: any;
```

IMCAOutBoundCommErrorResponsePayload

Response data payload object of OutboundCommError operation.

Attributes

result

Attribute which contains the result.

```
result: string;
```

error

Attribute which contains the error, if any.

```
error: any;
```

IMCAAgentStateEventResponsePayload

Response data payload object of AgentStateEvent operation.

Attributes

toolbarName

Attribute which contains the toolbar name.

```
toolbarName: string;
```

method

Attribute which contains method.

```
method: any;
```

uuid

Attribute which contains the uuid.

```
uuid: string;
```

result

Attribute which contains the result.

```
result: string;
```

error

Attribute which contains the error, if any.

```
error: any;
```

IMCAConfiguration

MCAConfiguration object of IMCAGetConfigurationResponsePayload.

Functions

faTrustToken

Attribute which contains the faTrustToken.

```
faTrustToken: string;
```

agentId

Attribute which contains agentId.

```
agentId: string;
```

agentPartyId

Attribute which contains the agentPartyId.

```
agentPartyId: string;
```

features

Attribute which contains the features.

```
features: string;
```

companionPanelUrl

Attribute which contains the companionPanelUrl.

```
companionPanelUrl: string;
```

companionPanelTitle

Attribute which contains the companionPanelTitle.

```
companionPanelTitle: string;
```

IMCAGetConfiguration

MCAGetConfiguration object of IMCAGetConfigurationResponsePayload.

Functions

getAgentId

Function to get agentId.

```
getAgentId(): string;
```

getAgentPartyId

Function to get agentPartyId.

```
getAgentPartyId(): string;
```

getFeatures

Function to get features.

```
getFeatures(): string[];
```

getCompanionPanelUrl

Function to get companionPanelUrl.

```
getCompanionPanelUrl(): string;
```

getCompanionPanelTitle

Function to get companionPanelTitle.

```
getCompanionPanelTitle(): string;
```

getFaTrustToken

Function to get faTrustToken.

```
getFaTrustToken(): string;
```

IMcaEventRequest

Generic event request object for MCA events.

Function

setAppClassification

Function to set appClassification.

```
setAppClassification(appClassification: string): void;
```

IMcaOnDataUpdatedEventResponse

Event response of OnDataUpdated MCA Event.

Function

getResponseData

Function to get responseData.

```
getResponseData(): IMcaOnDataUpdatedData;
```

IMcaOnDataUpdatedData

Event data of OnDataUpdated MCA Event.

Functions

getData

Function to get data.

```
getData: () => IMcaOnDataUpdatedEventData;
```

getOutData

Function to get outData.

```
getOutData: () => IMcaOnDataUpdatedOutData;
```

getEventId

Function to get eventId.

```
getEventId: () => string;
```

getOutputData

Function to get outputData.

```
 getOutputData: () => IMcaOnDataUpdatedOutDataResponse;
```

IMcaOnDataUpdatedOutDataResponse

Event data of OnDataUpdated response's outData object.

Function

getUpdateType

Function to get updateType.

```
getUpdateType: () => string;
```

IMcaOnDataUpdatedOutData

OutData object of OnDataUpdated event's response data.

Object Structure

The following is the object structure for IMcaOnDataUpdatedOutData:

```
interface IMcaOnDataUpdatedOutData {
 updateType: string;
 }
```

IMcaOnDataUpdatedEventData

Event data of OnDataUpdated event's response data.

Object Structure

The following is the object structure for IMcaOnDataUpdatedEventData:

```
interface IMcaOnDataUpdatedEventData extends IMcaEventData {
 outData: IMcaOnDataUpdatedOutData;
 eventId: string;
 eventSource: string;
 result: string;
 }
```

IMcaEventData

Generic event data of MCA event's response data.

Object Structure

The following is the object structure for IMcaEventData:

```
interface IMcaEventData {
 channel: string;
 channelType: string;
 method: string;
 toolbarName: string;
 uuid: string;
 }
```

IMcaonOutgoingEventResponse

Response of MCA OnOutgoingEvent.

Functions

getResponseData

Function to get response data.

```
getResponseData(): IMcaOnOutgoingEventData;
```

IMcaOnOutgoingEventData

Data in MCA OnOutgoingEvent Response.

Functions

getData

Function to get data from response object.

```
getData: () => IMcaOnOutgoingEventPayloadData;
```

getOutData

Function to get outData from response.

```
getOutData: () => IMcaOnOutgoingEventOutData;
```

getOutputData

Function to get full data in the event response.

```
getOutputData: () => IMcaOnOutgoingEventOutDataResponse;
```

IMcaOnOutgoingEventOutData

OutData object in MCA OnOutgoingEvent Response.

ObjectStructure

```
interface IMcaOnOutgoingEventOutData {
SVCMCA_ANI: string;
SVCMCA_DISPLAY_NAME: string;
SVCMCA_COMMUNICATION_DIRECTION: string;
SVCMCA_CALL_ID: string;
SVCMCA_CONTACT_ID: string;
SVCMCA_CONTACT_JOB_TITLE: string;
SVCMCA_INTERACTION_REF_OBJ_ID: string;
SVCMCA_INTERACTION_REF_OBJ_TYPE: string;
SVCMCA_ORG_ID: string;
```

```
}
```

IMcaOnOutgoingEventOutDataResponse

OutData Response object of OnOutGoingEvent Response object.

Functions

getSVCMCA_ANI

Function to get SVCMCA_ANI.

```
getSVCMCA_ANI: () => string;
```

getSVCMCA_DISPLAY_NAME

Function to get SVCMCA_DISPLAY_NAME.

```
getSVCMCA_DISPLAY_NAME: () => string;
```

getSVCMCA_COMMUNICATION_DIRECTION

Function to get SVCMCA_COMMUNICATION_DIRECTION.

```
getSVCMCA_COMMUNICATION_DIRECTION: () => string;
```

getSVCMCA_CALL_ID

Function to get SVCMCA_CALL_ID.

```
getSVCMCA_CALL_ID: () => string;
```

getSVCMCA_CONTACT_ID

Function to get getSVCMCA_CONTACT_ID.

```
getSVCMCA_CONTACT_ID: () => string;
```

getSVCMCA_ANI

Function to get SVCMCA_ANI.

```
getSVCMCA_ANI: () => string;
```

getSVCMCA_CONTACT_JOB_TITLE

Function to get getSVCMCA_CONTACT_JOB_TITLE.

```
getSVCMCA_CONTACT_JOB_TITLE: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_ID

Function to get getSVCMCA_INTERACTION_REF_OBJ_ID.

```
getSVCMCA_INTERACTION_REF_OBJ_ID: () => string;
```

getSVCMCA_INTERACTION_REF_OBJ_TYPE

Function to get getSVCMCA_INTERACTION_REF_OBJ_TYPE.

```
getSVCMCA_INTERACTION_REF_OBJ_TYPE: () => string;
```

getSVCMCA_ORG_ID

Function to get getSVCMCA_ORG_ID.

```
getSVCMCA_ORG_ID: () => string;
```

IMcaOnToolbarInteractionCommandEventResponse

Event Response object of OnToolbarAgentCommand Event

Functions

getResponseData

Function to get response data.

```
getResponseData(): IMcaOnToolbarAgentCommandDataResponse;
```

IMcaOnToolbarAgentCommandEventResponse

Event Response object of OnToolbarAgentCommand Event

Functions

getResponseData

Function to get response data.

```
getResponseData(): IMcaOnToolbarAgentCommandDataResponse;
```

IMcaOnToolbarInteractionCommandDataResponse

Data in OnToolbarAgentCommand event's response object.

Functions

getData

Function to get data object

```
getData(): IMcaOnToolbarInteractionCommandData;
```

getEventId

Function to get eventId.

```
getEventId(): string;
```

getCommand

Function to get the command.

```
getCommand(): string;
```

getSlot

Function to get the slot.

```
getCommand(): string;
```

getInData

Function to get inData.

```
getInData(): any;
```

getResult

Function to get the result.

```
getResult(): string;
```

getResultDisplayString

Function to get the resultDisplayString.

```
getResultDisplayString(): string;
```

getOutdata

Function to get outData.

```
getOutdata(): any;
```

setOutdata

Function to set outData.

```
setOutdata(data: any): void;
```

setResult

Function to setResult.

```
setResult(value: string): void;
```

IMcaOnToolbarAgentCommandDataResponse

Data in OnToolbarAgentCommand event's response object.

Functions

getData

Function to get data object

```
getData(): IMcaOnToolbarInteractionCommandData;
```

getEventId

Function to get eventId.

```
getEventId(): string;
```

getCommand

Function to get the command.

```
getCommand(): string;
```

getInData

Function to get inData.

```
getInData(): any;
```

getResult

Function to get the result.

```
getResult(): string;
```

getResultDisplayString

Function to get the resultDisplayString.

```
getResultDisplayString(): string;
```

getOutdata

Function to get outData.

```
getOutdata(): any;
```

setOutdata

Function to set outData.

```
setOutdata(data: any): void;
```

setResult

Function to setResult.

```
setResult(value: string): void;
```

IMcaOnToolbarInteractionCommandData

Data object in OnToolbarInteractionCommand event response.

Object structure

```
interface IMcaOnToolbarInteractionCommandData extends IMcaEventData {
 command: string;
 commandId: string;
 eventId: string;
 inData: any;
 outData: any;
 result: string;
 target: string;
 timeStamp: number;
 slot: string;
 resultDisplayString: string;
 }
```

IMcaOnToolbarAgentCommandDataResponse

Data in OnToolbarAgentCommand event's response object.

Functions

getData

Function to get data object

```
getData(): IMcaOnToolbarInteractionCommandData;
```

getEventId

Function to get eventId.

```
getEventId(): string;
```

getCommand

Function to get the command.

```
getCommand(): string;
```

getInData

Function to get inData.

```
getInData(): any;
```

getResult

Function to get the result.

```
getResult(): string;
```

getResultDisplayString

Function to get the resultDisplayString.

```
getResultDisplayString(): string;
```

getOutdata

Function to get outData.

```
getOutdata(): any;
```

setOutdata

Function to set outData.

```
setOutdata(data: any): void;
```

setResult

Function to setResult.

```
setResult(value: string): void;
```

IInteractionLogger

Functions

startLogging

This function can be executed from the interaction logger object to start logging interaction records. Internally this function will create a parent interaction record in SVC_INTERACTIONS table.

Once a parent interaction record is created in SVC_INTERACTIONS table, for each Fusion object (SR, Case) getting saved, an entry will be added to SVC_INTERACTION_REFS table. If a second object is saved, it will create a child interaction record in SVC_INTERACTIONS table and a new entry will be added to SVC_INTERACTIONS table with interaction id as the interaction id of child interaction got created. In other words, there will be a one-to-one mapping between interactions in SVC_INTERACTIONS table and interaction refs in SVC_INTERACTION_REFS table. This child interaction creation and interaction refs creation will continue until stopLogging is called. Also, it can be paused and resumed temporarily by calling pauseLogging and resumeLogging functions respectively.

Here's the syntax:

```
startLogging(interactionPayload?: Record<string, string>): Promise<string>;
```

Parameter Name	Required?	Description
interactionPayload	No	Additional information to be saved as part of parent interaction. For example, `'UefCustomTextTvm_c': 'Extensible Field Value 1'}`

Here's an example of the Typescript:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
```

Here's an example of the JavaScript:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
```

stopLogging

This function can be executed from the interaction logger object to stop logging interaction records. Internally this function will update the parent interaction record (StatusCd=ORA_SVC_CLOSED) (which was created as part of startLogging function) in SVC_INTERACTIONS table. If WrapUpPayload is provided, it will create a wrapUp record in SVC_MCA_INTERACT_SUMMARY table.

Here's the syntax:

```
startLogging(interactionPayload?: Record<string, string>, wrapUpPayload?: Record<string, string>): void;
```

Parameter Name	Required?	Description
interactionPayload	No	Additional information to be saved as part of parent interaction. For example, `'UefCustomTextTvm_c': 'Extensible Field Value 1'`
wrapUpPayload	No	If passed, it will create a WrapUp entry in the database. For example, `.stopLogging(null, {'InteractionNotes':'quick 123'}`

Here's an example of the Typescript:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
```

```
var parentInteractionId: string = await interactionLogger.startLogging();
interactionLogger.stopLogging(null, {'InteractionNotes':'quick 123'});
```

Here's an example of the JavaScript:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
interactionLogger.stopLogging();
```

pauseLogging

This function can be executed from the interaction logger object to pause logging interaction records.

Here's the syntax:

```
pauseLogging(): void;
```

Here's an example of the Typescript:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
```

Here's an example of the JavaScript:

```
var frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
```

resumeLogging

This function can be executed from the interaction logger object to resume an already paused interaction logging.

Here's the syntax:

```
resumeLogging(): void;
```

Here's an example of the Typescript:

```
var frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger: IInteractionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE',
 '300100572530155');
var parentInteractionId: string = await interactionLogger.startLogging();
await interactionLogger.pauseLogging();
await interactionLogger.resumeLogging();
```

Here's an example of the JavaScript:

```
vvar frameworkProvider = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app','v1')
var interactionLogger = frameworkProvider.getInteractionLogger('ORA_SVC_PHONE', '300100572530155');
var parentInteractionId = await interactionLogger.startLogging();
```

```
await interactionLogger.pauseLogging();
await interactionLogger.resumeLogging();
```

IInsightsContext

IInsightsContext is available under recordContext. IInsightsContext and provides the ability to listen to agent insights CCA's events and perform operations. For example, Publish new insights, Listening to Insights action trigger event and so on.

To get the reference to IInsightsContext, users can use the getInsightsContext function from recordContext. This function provides a reference to the insights.

Here's the syntax:

```
getInsightsContext(): Promise<IInsightsContext>;
```

Her's an example in Typescript:

```
/// <reference path="uiEventsFramework.d.ts"/>
const insightContext: IInsightContext = await recordContext.getInsightsContext();
```

Here's an example in JavaScript:

```
const insightContext = await recordContext.getInsightsContext();
```

Functions

subscribe

Use this function to get response data for for Set FieldValue subscription.

This API can be used to subscribe to events on insights CCA.

```
subscribe: (payload: IEventSubscriptionPayload, callbackFunction: (response:IEventResponsePayload) => void)
 => ISubscriptionContext;
```

Parameters

Parameter Name	Required?	Description
payload	Yes	The request object for the event to be subscribed to.
callbackFunction	Yes	A callback function, where we get the event response as its arguments.

The following code sample shows an example in Typescript to listen to the dismissal event from Insights, considering recordContext as current browser tabs active record:

```
let recordContext: IRecordContext = await
 uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext: IInsightContext = await recordContext.getInsightsContext();
const payload: IInsightsSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent');
payload.setId('insightsId1');
insightContext.subscribe(requestObject, (response: IInsightsDismissActionEventResponse ) => {
 console.log((response.getResponseData() as IInsightsDismissActionData).getInsightsId());
 console.log((response.getResponseData() as IInsightsDismissActionData).getReason());
});
```

The following code sample shows an example in JavaScript to listen to the dismissal event from Insights, considering recordContext as current browser tabs active record:

```
let recordContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent');
payload.setId('insightsId1');
insightContext.subscribe(requestObject, (response ) => {
 console.log(response.getResponseData().getInsightsId());
 console.log(response.getResponseData().getReason());
});
```

subscribeOnce

This API can be used to subscribe to a Fusion event just once. Using this API, external applications can listen to the application level events from the the Fusion application. But this subscription will be disposed of automatically after the first notification.

Here's the syntax:

```
subscribeOnce: (payload: IEventSubscriptionPayload, callbackFunction: (response: IEventResponsePayload) =>
 void => ISubscriptionContext;
```

Parameters

Parameter Name	Required?	Description
payload	Yes	The request object for the event to be subscribed to.
callbackFunction	Yes	A callback function, where we get the event response as its arguments.

Here's a Typescript example:

```
let recordContext: IRecordContext = await
 uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext: IInsightContext = await recordContext.getInsightsContext();
const payload: IInsightsSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent');
payload.setId('insightsId1');
insightContext.subscribeOnce(requestObject, (response: IInsightsDismissActionEventResponse ) => {
 console.log((response.getResponseData() as IInsightsDismissActionData).getInsightsId());
 console.log((response.getResponseData() as IInsightsDismissActionData).getReason());
});
```

Here's a JavaScript example:

```
let recordContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusOnInsightsDismissActionEvent');
payload.setId('insightsId1');
insightContext.subscribeOnce(requestObject, (response ) => {
 console.log(response.getResponseData().getInsightsId());
 console.log(response.getResponseData().getReason());
});
```

publish

This API can inform the Insights CCA to operate an action. The API will return a promise to which we can add a then-catch block. The then block would return the operation's status and any data returned after the process is completed.

Here's the syntax:

```
ublish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

Here's a Typescript example for the publish API used to publish the insight. Considering recordContext as current browser tabs active record:

```
let recordContext: IRecordContext = await
 uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext: IInsightContext = await recordContext.getInsightsContext();
const payload: IShowInsightsRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('ShowInsights');
payload.setId('insightsId1');
payload.setTitle('Custom title');
insightContext.publish(payload).then((response: IInsightsOperationResponse) => {
 console.log((response.getResponseData() as IInsightsOperationResponseData).getInsightsId());
});
```

Here's a JavaScript example:

```
let recordContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext().getActiveRecord();
let insightContext = await recordContext.getInsightsContext();
const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('ShowInsights');
payload.setId('insightsId1');
payload.setTitle('Custom title');
insightContext.publish(payload).then((response) => {
 console.log(response.getResponseData().getInsightsId());
});
```

dispose

This API can dispose or unregister any subscriptions that's added for an event. By disposing the subscription, the API will not trigger any further notifications to the client side receiver.

Here's the syntax:

```
dispose:() => void;
```

getSupportedEvents

This method will return all the supported events on top of globalContext.

Here's the syntax:

```
getSupportedEvents(): string[];
```

getSupportedActions

This method will return all the supported actions on top of GlobalContext.

Here's the syntax:

```
getSupportedActions(): string[];
```

IInsightsSubscriptionRequest

Functions

setId

Function to set id of to the subscription request object.

```
setId(id: string): void;
```

IShowInsightsRequest

Functions

setId

Function to set id of to the publish request object.

```
setId(id: string): void;
```

setTitle

Function to set title of to the publish request object.

```
setTitle(title: string): void;
```

setMessage

Function to set message of to the publish request object.

```
setMessage(message: string): void;
```

setHeader

Function to set category of to the publish request object.

```
setHeader(category: string): void;
```

setIcon

Function to set icon of to the publish request object.

```
setIcon(icon: string): void;
```

setType

Function to set type of to the publish request object.

```
setType(type: string): void;
```

action

Function to set action of to the publish request object.

```
action: () => IInsightsActionRequest;
```

setHeight

Sets the height of the MCA toolbar in pixels.

```
setHeight: (width: Pixels) => void;
```

IInsightsActionRequest

Functions

setActionDetails

Function to set action details of to the action function in publish request object.

```
setActionDetails(name: string, id: string): void;
```

IDismissInsights

Functions

setId

Function to set id of to the publish request object.

```
setId(id: string): void
```

IInsightsActionEventResponse

Functions

getResponseData

Function to get response of subscription of on action trigger event.

```
getResponseData(): IInsightsActionData;
```

IInsightsActionData

Functions

getActionId

Function to get id of the action triggered subscription of on action trigger event.

```
getActionId(): string;
```

getActionName

Function to get name of the action triggered subscription of on action trigger event.

```
getActionName(): string;
```

getInsightsId

Function to get id of the insights of the action triggered subscription of on action trigger event.

```
getInsightsId(): string;
```

IInsightsDismissActionEventResponse

Functions

getResponseData

Function to get response of subscription of on dismissal event.

```
getResponseData(): IInsightsDismissActionData;
```

IInsightsDismissActionData

Functions

getInsightsId

Function to get id of the insights subscription of on dismissal event.

```
getInsightsId(): string;
```

getReason

Function to get reason of the action triggered subscription of on dismissal event.

```
getReason(): string;
```

IInsightsOperationResponse

Functions

getResponseData

Function to get Response data of publish operation of show Insights and dismiss Insights.

```
getResponseData(): IInsightsOperationResponseData;
```

IInsightsOperationResponseData

Functions

getInsightsId

Function to get insights id in the response of publish operation of show Insights and dismiss Insights.

```
getInsightsId(): string;
```

IInsightsGetAllOperationResponse

Functions

getResponseData

Function to get Response data of publish operation of getAllInsights

```
getResponseData(): IInsightsGetAllOperationResponseData;
```

IInsightsGetAllOperationResponseData

Functions

getInsights

Function to get Response data as array of insights from publish operation of getAllInsights.

```
getInsights(): any;
```

IToolbarContext

Functions

Publish

```
publish: (requestObject: IOperationRequest) => Promise<IOperationResponse>;
```

Here's an example in Typescript:

```
let frameworkProvider: IUiEventsFrameworkProvider;
let mcaContext: IMultiChannelAdaptorContext;
let phoneContext: IPhoneContext;
let toolBarContext: IToolbarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest: ISetToolbarPropertiesActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties') as
 ISetToolbarPropertiesActionRequest;
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((operationResponse: IOperationResponse) => {
 console.log(operationResponse);
```

```
 const setToolbarPropertiesActionResponse: ISetToolbarPropertiesActionResponse = operationResponse as
 ISetToolbarPropertiesActionResponse;
 const setToolbarPropertiesActionResponseData: ISetToolbarPropertiesActionResponseData =
 setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess: boolean = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

Here's an example in JavaScript:

```
let frameworkProvider;
let mcaContextl
let phoneContext;
let toolBarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties');
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((setToolbarPropertiesActionResponse) => {
 console.log(setToolbarPropertiesActionResponse);
 const setToolbarPropertiesActionResponseData = setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

ISetToolbarPropertiesActionRequest

Functions

setWidth

Sets the width of the MCA toolbar in pixels.

```
setWidth: (width: Pixels) => void;
```

setHeight

Sets the height of the MCA toolbar in pixels.

```
setHeight: (width: Pixels) => void;
```

Here's an example in Typescript:

```
let frameworkProvider: IUiEventsFrameworkProvider;
let mcaContext: IMultiChannelAdaptorContext;
let phoneContext: IPhoneContext;
let toolBarContext: IToolbarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest: ISetToolbarPropertiesActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties') as
 ISetToolbarPropertiesActionRequest;
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((operationResponse: IOperationResponse) => {
 console.log(operationResponse);
 const setToolbarPropertiesActionResponse: ISetToolbarPropertiesActionResponse = operationResponse as
 ISetToolbarPropertiesActionResponse;
 const setToolbarPropertiesActionResponseData: ISetToolbarPropertiesActionResponseData =
 setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess: boolean = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

Here's an example in JavaScript:

```
let frameworkProvider;
let mcaContextl
let phoneContext;
let toolBarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties');
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((setToolbarPropertiesActionResponse) => {
 console.log(setToolbarPropertiesActionResponse);
 const setToolbarPropertiesActionResponseData = setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess = setToolbarPropertiesActionResponse.isSuccess();
```

```
}).catch((error) => console.error(error));
```

ISetToolbarPropertiesActionResponse

Functions

getResponseData

```
getResponseData(): ISetToolbarPropertiesActionResponseData;
```

isSucccess

```
isSuccess(): boolean;
```

Here's an example in Typescript:

```
let frameworkProvider: IUiEventsFrameworkProvider;
let mcaContext: IMultiChannelAdaptorContext;
let phoneContext: IPhoneContext;
let toolBarContext: IToolbarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest: ISetToolbarPropertiesActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties') as
 ISetToolbarPropertiesActionRequest;
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((operationResponse: IOperationResponse) => {
 console.log(operationResponse);
 const setToolbarPropertiesActionResponse: ISetToolbarPropertiesActionResponse = operationResponse as
 ISetToolbarPropertiesActionResponse;
 const setToolbarPropertiesActionResponseData: ISetToolbarPropertiesActionResponseData =
 setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess: boolean = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

Here's an example in JavaScript:

```
let frameworkProvider;
let mcaContextl
let phoneContext;
```

```
let toolBarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties');
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((setToolbarPropertiesActionResponse) => {
 console.log(setToolbarPropertiesActionResponse);
 const setToolbarPropertiesActionResponseData = setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

ISetToolbarPropertiesActionResponseData

Properties

```
export interface ISetToolbarPropertiesActionResponseData {
 status: string;
 payload: any;
}
```

Here's an example in Typescript:

```
let frameworkProvider: IUiEventsFrameworkProvider;
let mcaContext: IMultiChannelAdaptorContext;
let phoneContext: IPhoneContext;
let toolBarContext: IToolbarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE') as IPhoneContext;
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest: ISetToolbarPropertiesActionRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties') as
 ISetToolbarPropertiesActionRequest;
```

```
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((operationResponse: IOperationResponse) => {
 console.log(operationResponse);
 const setToolbarPropertiesActionResponse: ISetToolbarPropertiesActionResponse = operationResponse as
 ISetToolbarPropertiesActionResponse;
 const setToolbarPropertiesActionResponseData: ISetToolbarPropertiesActionResponseData =
 setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess: boolean = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

Here's an example in JavaScript:

```
let frameworkProvider;
let mcaContextl
let phoneContext;
let toolBarContext;

// init the frameworkProvider and the required contexts
try {
 frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
 mcaContext = await frameworkProvider.getMultiChannelAdaptorContext();
 phoneContext = await mcaContext.getCommunicationChannelContext('PHONE');
 toolBarContext = phoneContext.getToolbarContext();
} catch (e) {
 console.error(e);
}

// setup request object
const setToolbarPropertiesRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetToolbarProperties');
setToolbarPropertiesRequest.setHeight(200);
setToolbarPropertiesRequest.setWidth(300);

// publish the request on toolbarContext
toolBarContext.publish(setToolbarPropertiesRequest).then((setToolbarPropertiesActionResponse) => {
 console.log(setToolbarPropertiesActionResponse);
 const setToolbarPropertiesActionResponseData = setToolbarPropertiesActionResponse.getResponseData();
 const isSuccess = setToolbarPropertiesActionResponse.isSuccess();
}).catch((error) => console.error(error));
```

ISetChannelAvailabilityRequest

Functions

setChannel

Enables or disables the phone icon.

```
setAvailability: (isAvailable: boolean) => void;
```

Here's an example in Typescript:

```
// init framework and get globalContext
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

// create request object
const request: ISetChannelAvailabilityRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability') as
 ISetChannelAvailabilityRequest;
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response: IOperationResponse) => {
 const isSuccess: boolean = (response as ISetChannelAvailabilityResponse).isSuccess();
 if (!isSuccess) {
 console.log('error', (response as ISetChannelAvailabilityResponse).getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

Here's an example in JavaScript:

```
// init framework and get globalContext
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();

// create request object
const request = frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability');
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response) => {
 const isSuccess = response.isSuccess();
 if (!isSuccess) {
 console.log('error', response.getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

getError

Set the communication channel type, such as PHONE.

```
setChannel: (channel: string) => void;
```

Here's an example in Typescript:

```
// init framework and get globalContext
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

// create request object
const request: ISetChannelAvailabilityRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability') as
 ISetChannelAvailabilityRequest;
request.setAvailability(isAvailable);
```

```
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response: IOperationResponse) => {
 const isSuccess: boolean = (response as ISetChannelAvailabilityResponse).isSuccess();
 if (!isSuccess) {
 console.log('error', (response as ISetChannelAvailabilityResponse).getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

Here's an example in JavaScript:

```
// init framework and get globalContext
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();

// create request object
const request = frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability');
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response) => {
 const isSuccess = response.isSuccess();
 if (!isSuccess) {
 console.log('error', response.getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

ISetChannelAvailabilityResponse

Functions

isSuccess

```
isSuccess: () => boolean;]
```

Here's an example in Typescript:

```
// init framework and get globalContext
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

// create request object
const request: ISetChannelAvailabilityRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability') as
 ISetChannelAvailabilityRequest;
request.setAvailability(isAvailable);
request.setChannel(channelType);
```

```
// publish request on globalContext
globalContext.publish(request).then((response: IOperationResponse) => {
 const isSuccess: boolean = (response as ISetChannelAvailabilityResponse).isSuccess();
 if (!isSuccess) {
 console.log('error', (response as ISetChannelAvailabilityResponse).getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

Here's an example in JavaScript:

```
// init framework and get globalContext
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();

// create request object
const request = frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability');
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response) => {
 const isSuccess = response.isSuccess();
 if (!isSuccess) {
 console.log('error', response.getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

getError

```
getError: () => any;
```

Here's an example in Typescript:

```
// init framework and get globalContext
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

// create request object
const request: ISetChannelAvailabilityRequest =
 frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability') as
 ISetChannelAvailabilityRequest;
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response: IOperationResponse) => {
 const isSuccess: boolean = (response as ISetChannelAvailabilityResponse).isSuccess();
 if (!isSuccess) {
 console.log('error', (response as ISetChannelAvailabilityResponse).getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

Here's an example in JavaScript:

```
// init framework and get globalContext
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID''v1');
const globalContext = await frameworkProvider.getGlobalContext();

// create request object
const request = frameworkProvider.requestHelper.createPublishRequest('SetChannelAvailability');
request.setAvailability(isAvailable);
request.setChannel(channelType);

// publish request on globalContext
globalContext.publish(request).then((response) => {
 const isSuccess = response.isSuccess();
 if (!isSuccess) {
 console.log('error', response.getError())
 } else {
 console.log('success');
 }
}).catch(e => console.log(e));
```

6 MCA integration

Overview of integrating MCA with different suppliers

Required Components

This integration requires the following components:

- The Fusion application.
- The Media toolbar
  - Acts as a placeholder for rendering your CTI application.
  - The URL of the partner CTI application can be configured through Fusion configuration management screen. Once this configuration is set, this URL is rendered in an iFrame in the partner application view.
  - The Media toolbar enables communication between the Fusion application and the CTI supplier
  - The communication between the Fusion application and the partner application is enabled through a JavaScript library which can be used by the partner application.
- Third party CTI suppliers.
  - You can choose your own supplier for the integration.

While integrating a 3rd party supplier, Fusion CTI provides a library having 2 types of APIs:

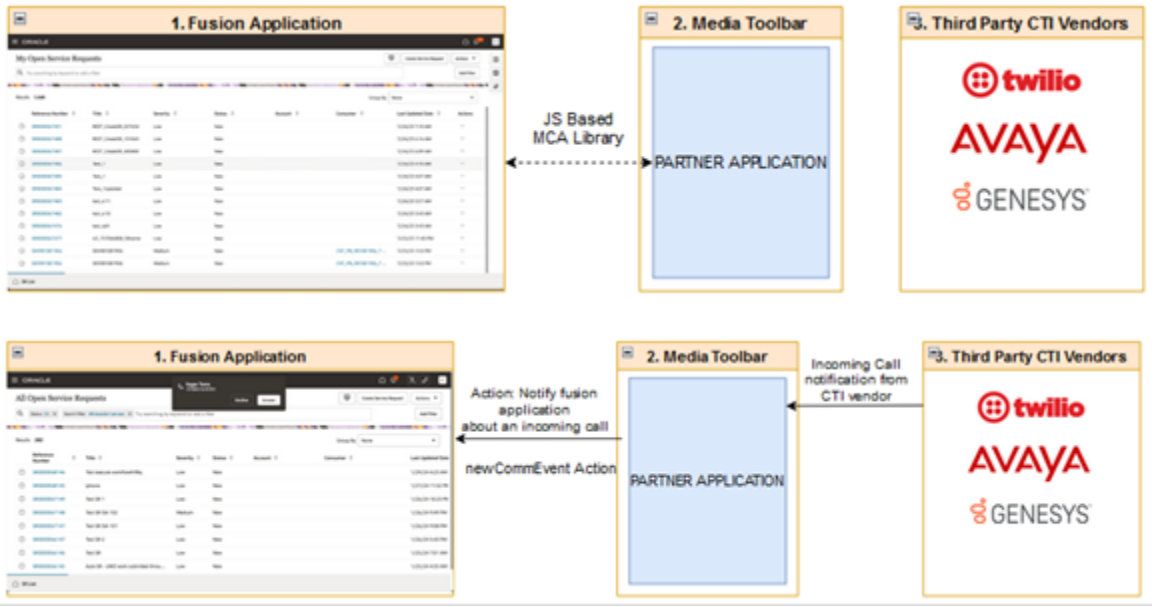
- Actions
- Events

General steps to execute actions and events APIs:

- Get the proper context.
- Create the request object.
- Call the API.

Actions

You use Action APIs when a partner application wants the Fusion application to perform certain operations. For example, a Partner application notifies the Fusion application to render an incoming call dialog when an incoming call is received, the NewCommEvent action.



Supported Actions

Action	Description
agentStateEvent	This action can be fired from the partner application to make the current logged in agent ready to make or accept calls. This will make the phone icon enabled in the Fusion application.
newCommEvent	When an incoming call is received from the CTI supplier to the partner application, the newCommEvent action is fired from the partner application to inform the Fusion application about the incoming call. The the Fusion application will render a dialog box with the name and incoming phone number.
startCommEvent	When the CTI supplier notifies the partner application that a call is accepted, the partner application should fire the startCommEvent action to inform Fusion application that the call is accepted.
closeCommEvent	The partner application can fire this action whenever the call is disconnected from the CTI supplier or from the Fusion application by the agent.
outboundCommErro	The partner application can fire this action to notify the Fusion application that an error occurred during initiation of the outbound event.
getConfiguration	The partner application can fire this action to get configuration information that enables the toolbar to evaluate the features supported by the Fusion application.
disableFeature	Informs the Fusion application that a subset of available functionality must be disabled because the toolbar hasn't implemented it.
readyForOperation	Notifies the Fusion application that the toolbar is ready for the operation.

Here's example code showing the firing of actions:

```
// Step 1: Get the proper context
```

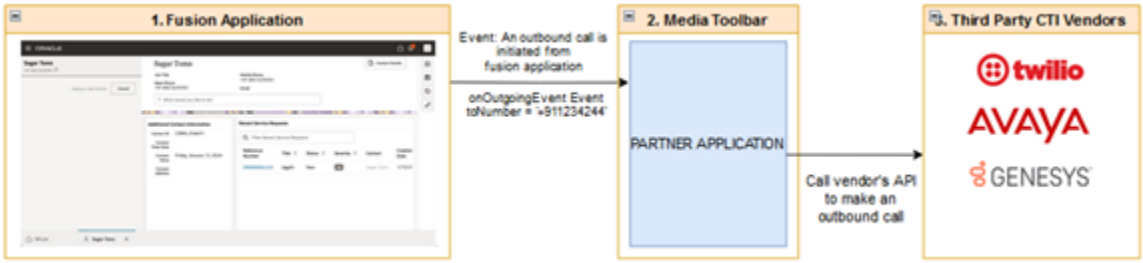
```
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;

// Step 2: Create the request object
const request: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('<Action Name>');
// Set request object properties

// Step 3: Invoke the API
phoneContext.publish(request).then((res: IOperationResponse) => {
 // response after firing the action
}).catch(() => {
})
```

Events

Whenever an agent performs an action on the Fusion application, the Fusion application will raise some events through the MCA library and these events can be listened by the partner application.

Use Events APIs whenever the partner application needs to listen to the actions performed by the agent in the Fusion application.

For example, the Fusion application raises an event when an outbound call is begun from the Fusion application – onOutgoingEvent.



Supported Events

Event	Description
onToolbarInteractionCommand	Once a call is received in the Fusion application, the agent can accept, reject or disconnect it. onToolbarInteractionCommand is fired from the Fusion application based on the agent's action (accept, reject or disconnect). • **accept**: onToolbarInteractionCommand is fired with command as **accept** when the agent accepts the call from the Fusion application. • **reject**: onToolbarInteractionCommand is fired with command as **reject** when the agent declines the call from the Fusion application. • **disconnect**: onToolbarInteractionCommand is fired with command as **disconnect** when the agent hangup the call from the Fusion application.
onDataUpdated	This event is used to listen to a DataUpdate event that happens in Fusion application.

Event	Description
onOutgoingEvent	Whenever an outbound call is started from the Fusion application, this event will gets fired and the partner application can listen to this event and inform the CTI supplier to make the outbound call.

Here's an example of subscribing to any of the events:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;

// Step 2: Create the request object
const request: IMcaEventRequest = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('<Event
 Name>');
// Set request object properties

// Step 3: Invoke the API
phoneContext.subscribe(request, (response: IEventResponse) => {
 // subscription response
});
```

CTI scenarios

Here are three basic scenarios for the CTI application.

1. Make the agent available to make or accept calls.
2. Handle the incoming calls.
3. Handle the outbound calls.

How do I initialize the partner application as an MCA application?

Before implementing these scenarios, you must initialize the partner application as an MCA application to enable the communication with the Fusion application. Here's how you do it:

1. Inject the UIEF (ui-events-framework) JavaScript library into the partner application. To do this, you must include the ui-events-framework-library into the partner application's HTML page. Here's the code to inject the library to the application:
   ```
   // Step 1: Include the javascript library in partner application
   <script src="https://static.oracle.com/cdn/ui-events-framework/libs/ui-events-framework-client.js"></
   script>
   ```
2. Initialize the UIEF library. Once you can inject the UIEF JavaScript file into the HTML. You can initialize the framework using the following line:
   ```
   // Step 2: Initialize the library from partner application
   const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
    CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('PARTNER APP NAME', 'v1');
   ```

Make an agent available to make or accept calls

Once the agent is logged in to the Fusion application, they can make themselves available to accept or make phone calls.

To enable this, you use the agentStateEventOperation. The following code can be executed from the partner application to make the agent available:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await multiChannelAdaptorContext.getCommunicationChannelContext('PHONE')
 as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaAgentStateEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('agentStateEventOperation') as
 IMcaAgentStateEventActionRequest;
// Set request object properties
requestObject.setEventId('1');
requestObject.setIsAvailable(true);
requestObject.setIsLoggedIn(true);
requestObject.setState('AVAILABLE');
requestObject.setStateDisplayString('Idle');

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaAgentStateEventActionResponse) => {
 console.log('AGENT IS READY', operationResponse);
}).catch(() => {
})
```

One this API is executed, the Phone icon becomes available in the UI.

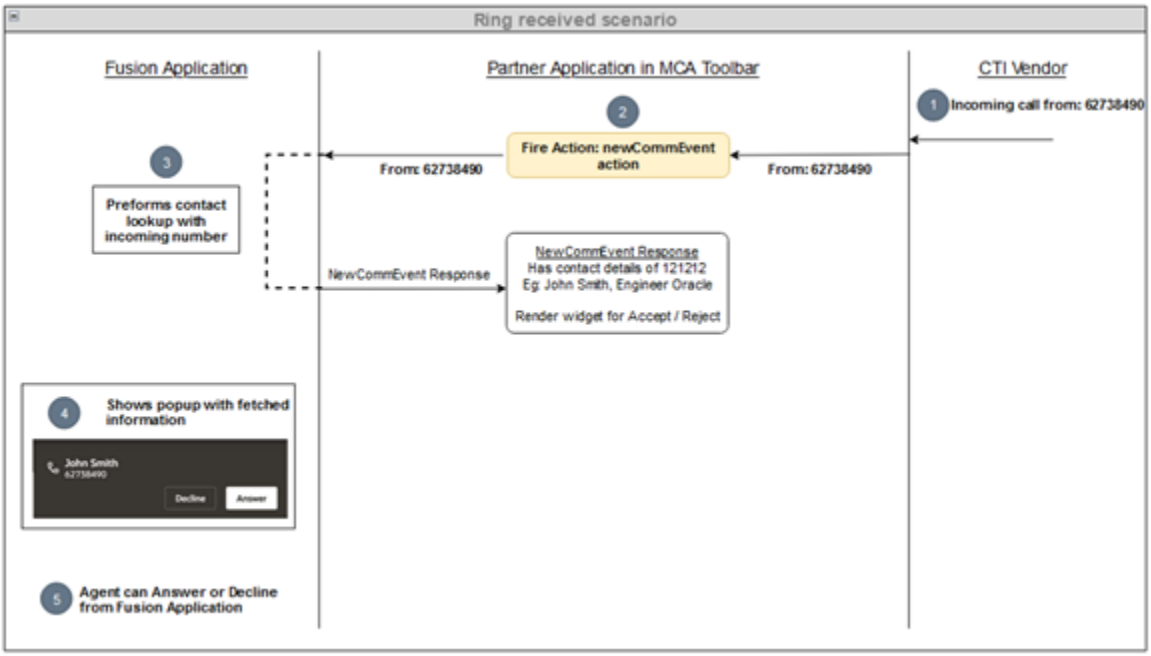
Handle incoming calls

This scenario can divided into four states.

1. Call received
2. Call accepted from Fusion
3. Call rejected from Fusion
4. Call disconnected from Fusion

Call received

During this state, a customer is calling a service center number and phone line is ringing.



This scenario goes this way:

1. The call is first received by the CTI supplier which notifies the partner application that there's an incoming call.
2. The partner application notifies the Fusion application of the incoming call by firing the `newCommEvent` action.

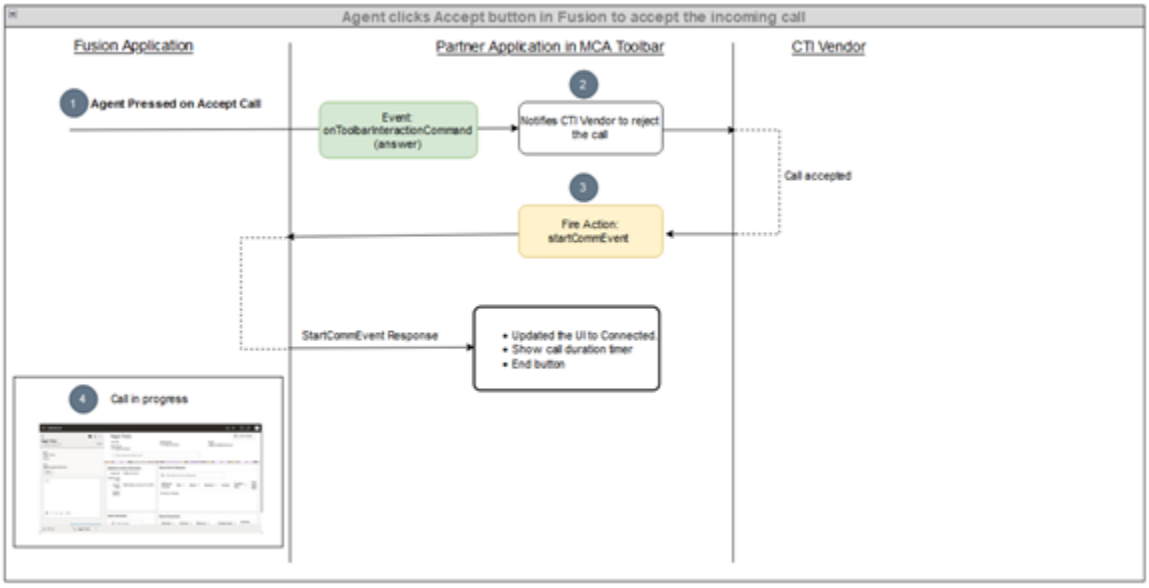
   The following code can be executed from the partner application to fire a `newCommEvent` action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent') as
 IMcaNewCommEventActionRequest;
// Set request object properties
requestObject.setEventId('1');
requestObject.getInData().setInDataValueByAttribute('SVCMCA_ANI', phoneNumber);
requestObject.getInData().setInDataValueByAttribute("SVCMCA_COMMUNICATION_DIRECTION",
 "ORA_SVC_INBOUND");
requestObject.setAppClassification('ORA_SERVICE');

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaNewComActionResponse) => {
 // Custom logic for handling the partner application UI
}).catch(() => {
})
```

3. Once the Fusion application identifies this action, it performs the contact number lookup and return the contact details in the action response.
4. Finally the Fusion application notifies the agent about the incoming call through a dialog box.
5. At this point, the agent can either answer or decline the call.

Call accepted from Fusion

In this scenario the agent clicks the Answer button, accepting the call.



1. When the agent clicks the Answer button in the Fusion application, the `onToolbarInteractionCommand` event is fired with the command as accept.

   Here's the code to listen to the `onToolbarInteractionCommand` event. This code can be executed from the partner application to subscribe to `onToolbarInteractionCommand` event:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand') as
 IMcaEventRequest;

// Step 3: Invoke the API
phoneContext.subscribe(requestObject, (eventResponse: IMcaOnToolbarInteractionCommandEventResponse) => {
 const eventResponseDetails: IMcaOnToolbarInteractionCommandDataResponse =
 eventResponse.getResponseData();
 const command: string = eventResponseDetails.getCommand();
 switch (command) {
 case "accept":
 // Notify CTI Vendor to accept the call
 break;
 case "disconnect":
 // Notify CTI Vendor to disconnect the call
 break;
 case "reject":
 // Notify CTI Vendor to disconnect the call
 break;
 }
}).catch(() => {
})
```

2. The partner application receives this event and if the event is to accept a call, the partner application notifies the CTI supplier to accept the call.
3. Once the CTI supplier notifies the partner application that the call is accepted, the partner application can fire the `startCommEvent` action.

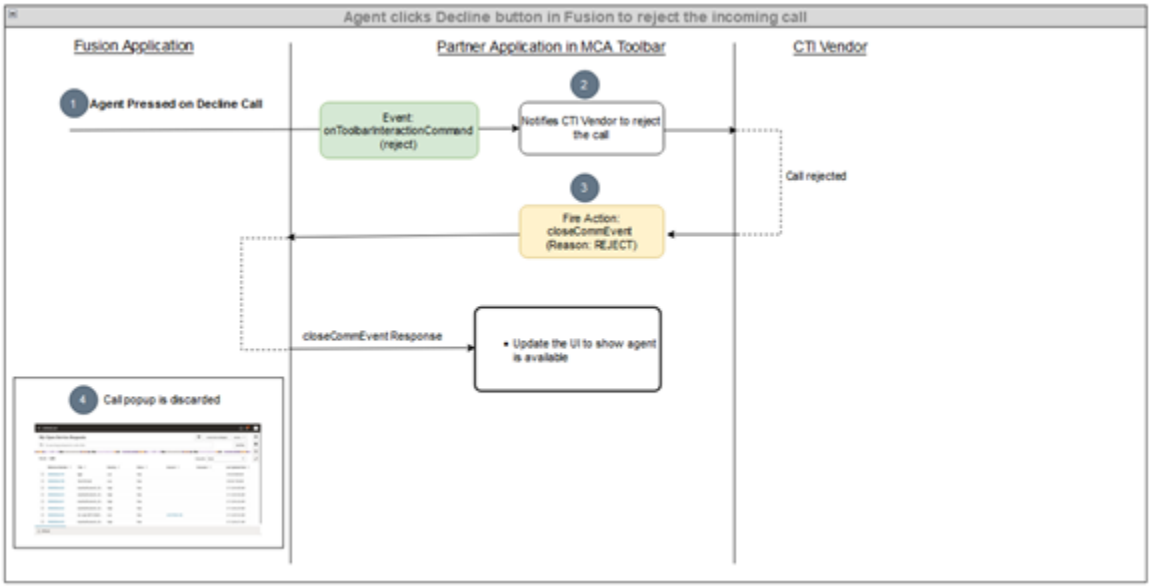
   The following code can be executed from the partner application to fire a `startCommEvent` action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
request.setAppClassification('ORA_SERVICE');
request.setEventId('1');

// Step 3: Invoke the API
phoneContext.publish(request).then((operationResponse: IMcaStartComActionResponse) => {
 // Extract the required data from response and use for updating the partner app
 const contactName: string = operationResponse.getResponseData().getData()['SVCMCA_CONTACT_NAME']; //
 sample for getting the contact name
}).catch(() => {
});
```

4. Once the Fusion application identifies this action, the matched contact and an engagement panel will be opened based on the rules configured in the Fusion configuration management for the screen pop.

Call rejected from Fusion

In this scenario, the agent clicks the Decline button to reject the call.



1. When the agent clicks the Answer button in the Fusion application, the `onToolbarInteractionCommand` event is fired with command as reject.

   The following code can be executed from the partner application to subscribe to the `onToolbarInteractionCommand` event:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand') as
 IMcaEventRequest;

// Step 3: Invoke the API
phoneContext.subscribe(requestObject, (eventResponse: IMcaOnToolbarInteractionCommandEventResponse) => {
 const eventResponseDetails: IMcaOnToolbarInteractionCommandDataResponse =
 eventResponse.getResponseData();
 const command: string = eventResponseDetails.getCommand();
 switch (command) {
 case "accept":
 // Notify CTI Vendor to accept the call
 break;
 case "disconnect":
 // Notify CTI Vendor to disconnect the call
 break;
 case "reject":
 // Notify CTI Vendor to disconnect the call
 break;
 }
}).catch(() => {
})
```

2. The partner application receives this event and if the event is to reject a call, the partner application notifies the CTI supplier to reject the call.

3. Once the CTI supplier notifies the partner application that the call is rejected, the partner application can fire the `closeCommEvent` action with reason as REJECT.

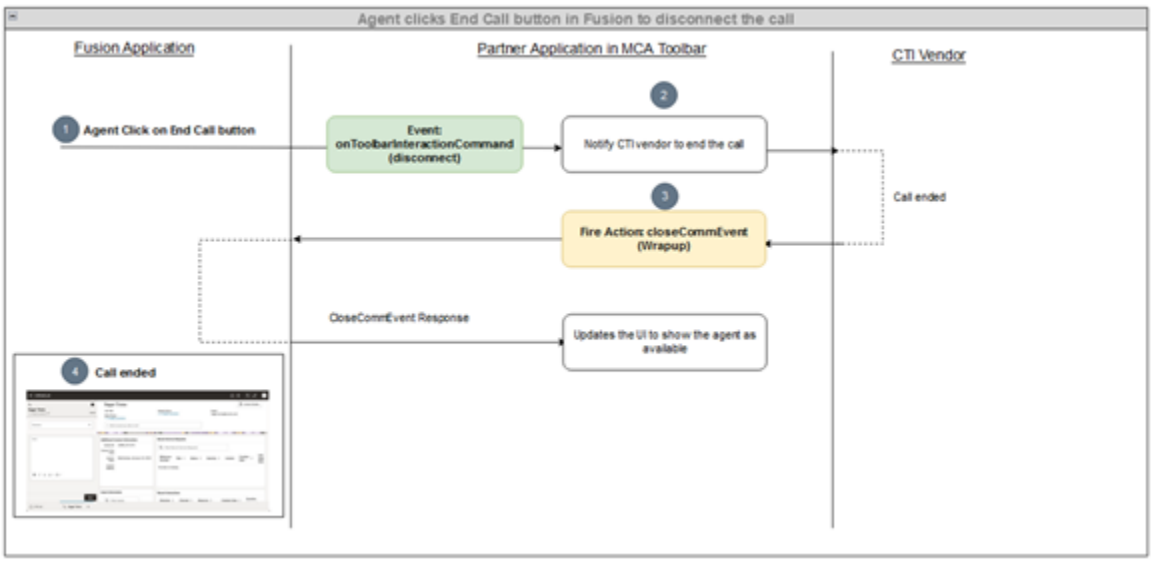
   The following code can be executed from the partner application to fire a `closeCommEvent` action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent') as
 IMcaCloseCommEventActionRequest;
// Set request object properties, in this scenario we need to set the reason as reject
requestObject.setReason("REJECT");

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaCloseComActionResponse) => {
 console.log('closeCommEvent fired', operationResponse);
}).catch(() => {
})
```

4. Once the fusion application identifies this action, the call dialog box will be discarded from the UI.

Call disconnected from Fusion

In this scenario, the agent clicks the End Call button in Fusion when the conversation is complete.



1. When the agent clicks the End Call button in the Fusion application, the `onToolbarInteractionCommand` event is fired with the disconnect command.

   The following code can be executed from the partner application to subscribe to the `onToolbarInteractionCommand` event:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
```

```
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand') as
 IMcaEventRequest;

// Step 3: Invoke the API
phoneContext.subscribe(requestObject, (eventResponse: IMcaOnToolbarInteractionCommandEventResponse) => {
 const eventResponseDetails: IMcaOnToolbarInteractionCommandDataResponse =
 eventResponse.getResponseData();
 const command: string = eventResponseDetails.getCommand();
 switch (command) {
 case "accept":
 // Notify CTI Vendor to accept the call
 break;
 case "disconnect":
 // Notify CTI Vendor to disconnect the call
 break;
 case "reject":
 // Notify CTI Vendor to disconnect the call
 break;
 }
}).catch(() => {
})
```

2. The partner application receives this event and if the event is to disconnect a call, the partner application notifies the CTI supplier to disconnect the call.
3. Once the CTI supplier notifies the partner application that the call is disconnected, the partner application can fire the `closeCommEvent` action.

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent') as
 IMcaCloseCommEventActionRequest;
// Set request object properties
requestObject.setReason("WRAPUP");

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaCloseComActionResponse) => {
 console.log('closeCommEvent fired', operationResponse);
}).catch(() => {
})
```

4. Once the Fusion application identifies this action, it renders the wrap up window in the UI.

Handle outbound calls

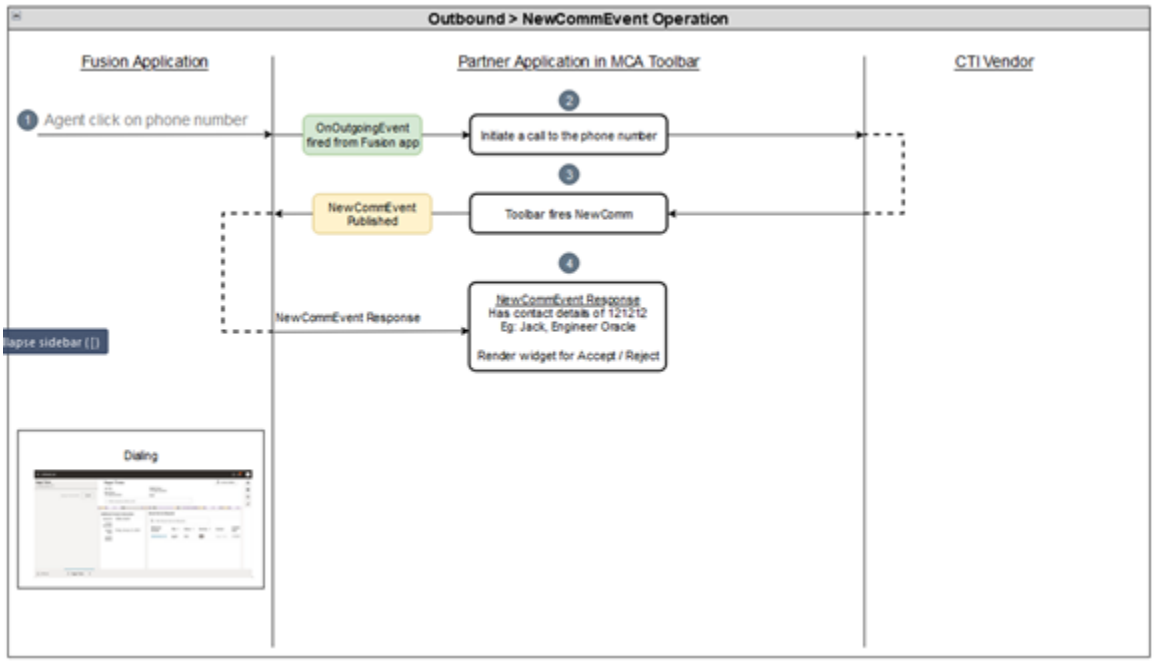
This scenario can be divided into four states.

1. Start outbound call
2. Call accepted by the customer
3. Call disconnected by the agent or customer

4. Call declined by the customer

Start outbound call

The agent clicks the phone number from the Fusion application and the call state is ringing.



1. The agent starts an outbound call from the Fusion application by clicking on the phone number, launching the onOutgoingEvent event.

   The following code can be executed from the partner application to subscribe to onOutgoingEvent event:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onOutgoingEvent') as
 IMcaEventRequest;
requestObject.setAppClassification('ORA_SERVICE');

// Step 3: Invoke the API
phoneContext.subscribe(requestObject, async (eventResponse: IMcaonOutgoingEventResponse) => {
 // Use the event response to get the phone number and invoke the initiate call api of CTI vendor
});
```

2. The partner application listening for this event then notify the CTI supplier to start the call.
3. The partner application will notify the call initiation to the Fusion application by publishing the newCommEvent action.

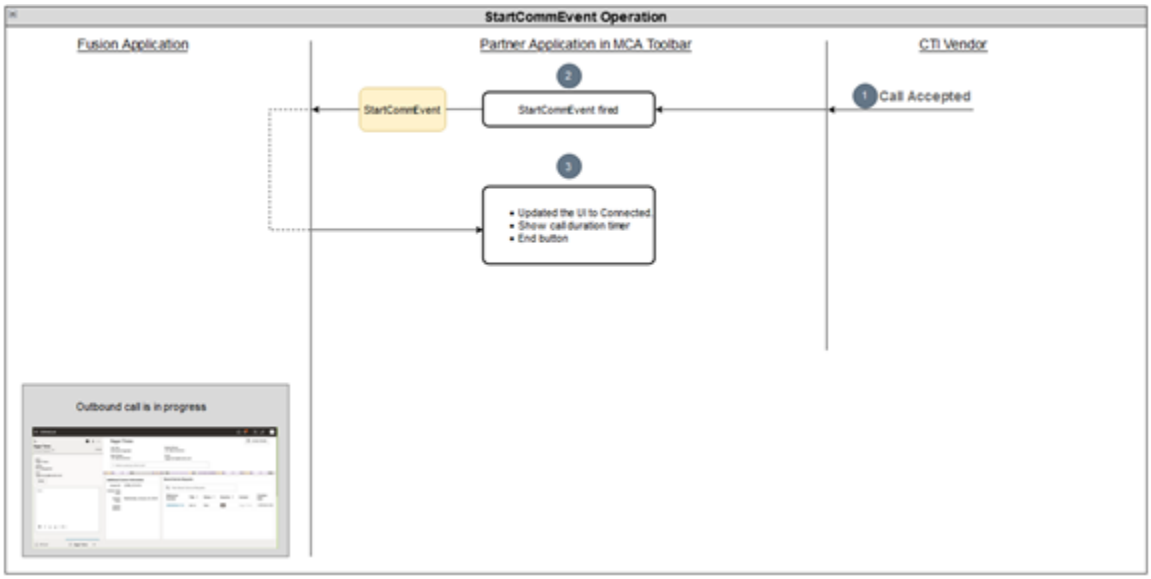
   The following code can be executed from the partner application to fire a newCommEvent action:

```
// Step 1: Get the proper context
```

```
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('newCommEvent') as
 IMcaNewCommEventActionRequest;
// Set request object properties
requestObject.setEventId('1');
requestObject.getInData().setInDataValueByAttribute('SVCMCA_ANI', phoneNumber);
requestObject.getInData().setInDataValueByAttribute("SVCMCA_COMMUNICATION_DIRECTION",
 "ORA_SVC_OUTBOUND");
requestObject.setAppClassification('ORA_SERVICE');

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaNewComActionResponse) => {
 // Handle the custom logic for UI updates here
}).catch(() => {
})
```

4. The action response from fusion application has the contact details which can be used by the Partner application for rendering the UI.

Call accepted by a customer



1. The Customer accepts the incoming call and the CTI supplier notifies the partner application. The partner application publishes the startCommEvent action to notify the Fusion application.

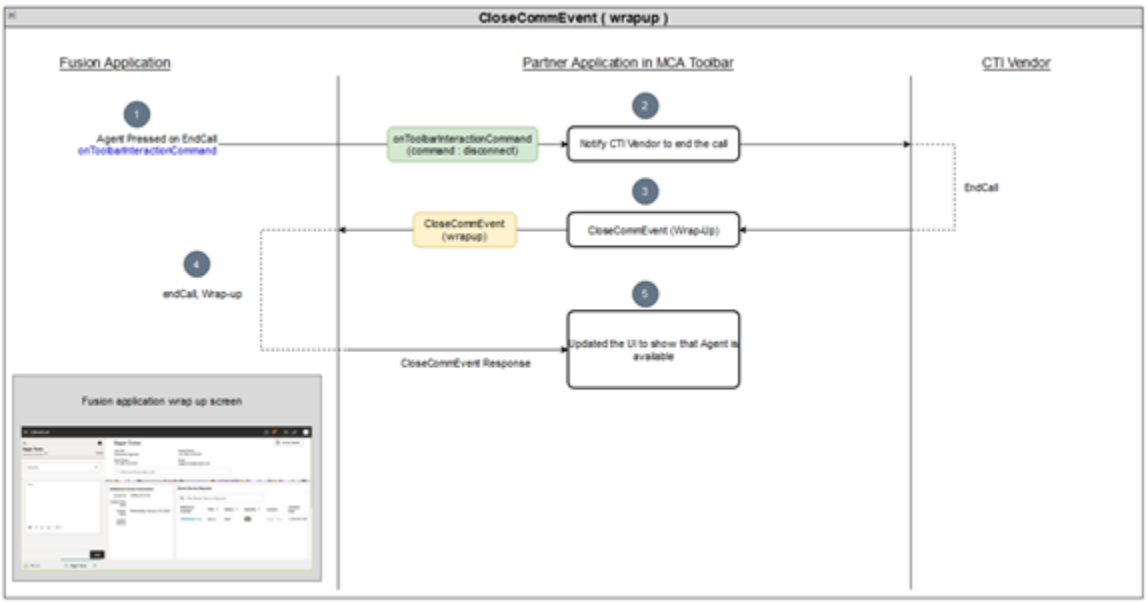
   The following code can be executed from the partner application to fire a startCommEvent action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
```

```
const request: IMcaStartCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('startCommEvent') as
 IMcaStartCommEventActionRequest;
request.setAppClassification('ORA_SERVICE');
request.setEventId('1');

// Step 3: Invoke the API
phoneContext.publish(request).then((operationResponse: IMcaStartComActionResponse) => {
 // Extract the required data from response and use for updating the partner app
 const contactName: string = operationResponse.getResponseData().getData()['SVCMCA_CONTACT_NAME']; //
 sample for getting the contact name
}).catch(() => {
});
```

2. On completing the startCommEvent the partner application can update the UI as required to show the in-progress call.

Call disconnected by the agent or customer



1. The agent ends the call from the Fusion application which fires onToolbarInteractionCommand Event with command as disconnect

   The following code can be executed from the partner application to subscribe to onToolbarInteractionCommand event:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('onToolbarInteractionCommand') as
 IMcaEventRequest;

// Step 3: Invoke the API
phoneContext.subscribe(requestObject, (eventResponse: IMcaOnToolbarInteractionCommandEventResponse) => {
```

```
 const eventResponseDetails: IMcaOnToolbarInteractionCommandDataResponse =
 eventResponse.getResponseData();
 const command: string = eventResponseDetails.getCommand();
 switch (command) {
 case "accept":
 // Notify CTI Vendor to accept the call
 break;
 case "disconnect":
 // Notify CTI Vendor to disconnect the call (In this use-case your logic should be here)
 break;
 case "reject":
 // Notify CTI Vendor to reject the call
 break;
 }
}).catch(() => {
})
```

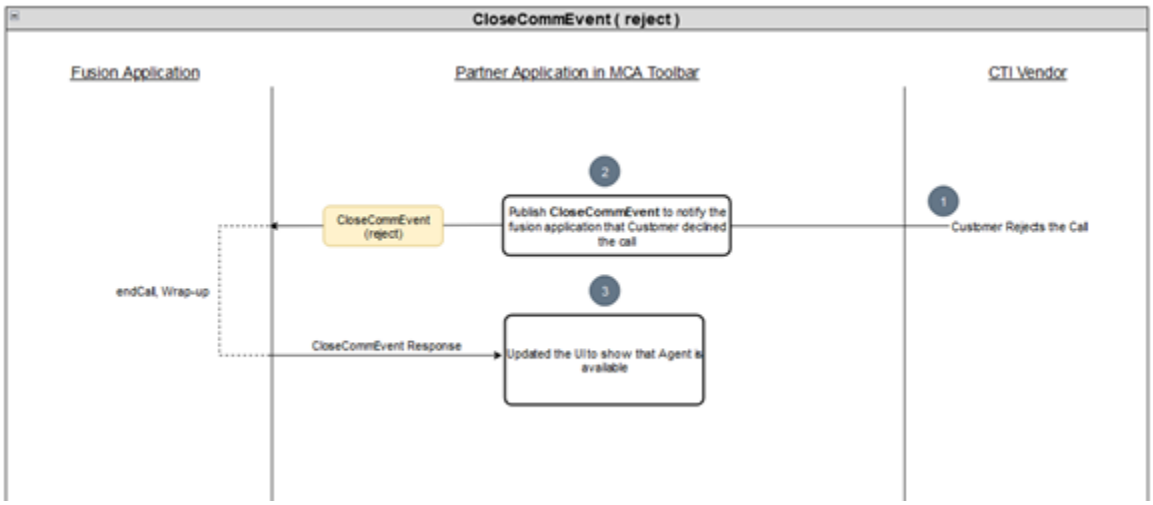
2. The Partner application listening for this event will handle the event by notifying the CTI supplier to disconnect the call.
3. Once the call is disconnected the partner application publishes closeCommEvent to the fusion application with reason as a wrap up.

   The following code can be executed from the partner application to fire a closeCommEvent action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent') as
 IMcaCloseCommEventActionRequest;
// Set request object properties, in this scenario we need to set the reason as Wrapup
requestObject.setReason("WRAPUP");

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaCloseComActionResponse) => {
 console.log('closeCommEvent fired', operationResponse);
}).catch(() => {
})
```

4. The Fusion application performs the wrap up flow and displays a note.
5. On the CloseCommEvent response the partner application updates the UI as required.

Call declined by the customer



1. The customer rejects the incoming call and the CTI supplier notifies the partner application.
2. The partner application updates the UI and publish the closeCommEvent to the Fusion application.

   The following code can be executed from the partner application to fire a closeCommEvent action:

```
// Step 1: Get the proper context
const multiChannelAdaptorContext: IMultiChannelAdaptorContext = await
 uiEventsFrameworkInstance.getMultiChannelAdaptorContext();
const phoneContext: IPhoneContext = await
 multiChannelAdaptorContext.getCommunicationChannelContext('PHONE') as IPhoneContext;

// Step 2: Create the request object
const requestObject: IMcaNewCommEventActionRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('closeCommEvent') as
 IMcaCloseCommEventActionRequest;
// Set request object properties, in this scenario we need to set the reason as reject
requestObject.setReason("REJECT");

// Step 3: Invoke the API
phoneContext.publish(requestObject).then((operationResponse: IMcaCloseComActionResponse) => {
 console.log('closeCommEvent fired', operationResponse);
}).catch(() => {
})
```

3. The partner application updates the UI to show that the agent is available.