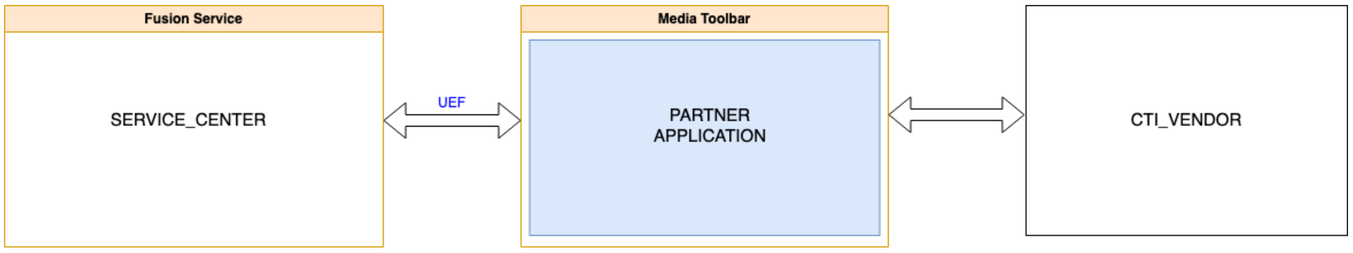
1. How to use UEF-MCA APIs for implementing the telephony events in Service Center with the Redwood User Experience

Overview

Computer Telephony Integration (CTI) enables integration of a telephony system with Fusion Service or Fusion Sales applications. You can load a PARTNER telephony system into a toolbar. By integrating the Fusion application with a CTI telephony system provider, an agent can make an outbound call or accept an inbound call from the Fusion application itself.

To connect with telephony partners, go to Oracle Cloud Marketplace and search for any one of the following:

- OpenMethods PopFlow
- OpenMethods Harmony (choose Harmony for B2B Service)
- Five9 (Cloud Contact Center for Oracle B2B Service)



How is this document organized?

This document is designed to be viewed from a PARTNER point of view.

The PARTNER is the intermediary, who handles the communication between SERVICE_CENTER & CTI_VENDOR. This document covers a generic approach of how to use UEF APIs for telephony event/action between SERVICE_CENTER & PARTNER. The Generic Events mentioned in the following tables are listened by the PARTNER application. The source of the event is either SERVICE_CENTER or the CTI_VENDOR.

An introduction to UI Events Framework (UEF)

UI Events Framework (UEF) is a library that allows third party applications to interact with Service Center at the UI level. It can listen to events generated from Service Center and performs actions to the same. To give an example, an input field value in the UI can be set with UEF from an application loaded into the toolbar. This is termed as an action in UEF. We can subscribe to a field value change using UEF so that when the input value is manually changed, an event is triggered in service center and our subscription gets notified accordingly. Aforesaid field level functionalities are some of the very basic features of UEF. It can do a variety of complex functionalities like open a service request in a new tab and perform field level operations there, close it from another tab, perform operations related to side pane, open modal, show notification, and many more.

UEF follows a standard pattern in all of its APIs. You can see this in the code snippets given below.

UEF's pattern	Example

1. Create a request object
2. Set the request object with required data
3. Perform publish or subscribe at a specific context
	`const requestObject = frameworkProvider.requestHelper.createPublishRequest ('cxEventBusSetFieldValueOperation');`
	`requestObject.field().setValue('ServiceRequest.Title', 'New Title');`
	`await recordContext.publish(requestObject)`

UEF's Media toolbar APIs (UEF-MCA APIs) are a set of JavaScript functions to enable a partner to integrate their media toolbars with ServiceCenter. These APIs facilitate communication between the media toolbar and ServiceCenter to exchange the configuration information, events & payload data in response to the inbound or outbound interaction requests.

CTI Generic Events and Actions Mapping to MCA (Multi Channel Architecture) APIs

The tables given below (*CTI Generic Events and Actions Mapping Table*) in the next section will act as the single point of reference for anyone including a non technical person to understand a generic telephony event/action and how to use MCA APIs for the same. This table will also attach sample code snippet for those generic telephony events/actions.

The table has the following columns:

- Generic Event
- Event Origin
- Event Description
- Expected Action
- UEF API to achieve Expected Action

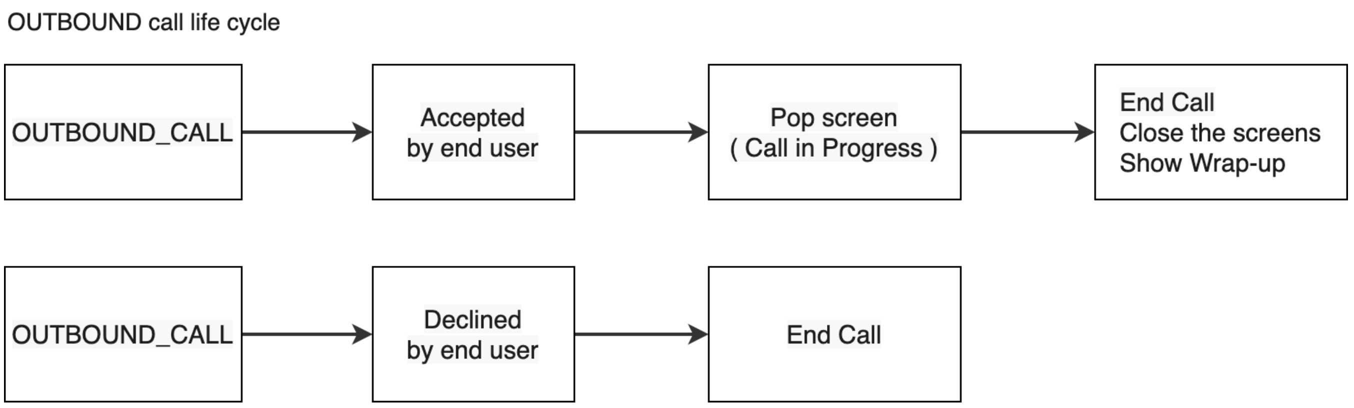
The Generic Event column is the reader's starting point. Take an INBOUND call scenario as an example where the payload is received at the PARTNER from the CTI_VENDOR.

The reader has to start reading the table from the '*Generic Event*' column. As an example, consider an inbound call scenario where the payload is received at the toolbar application from the CTI_VENDOR. In this case, the Generic Event is RING_RECEIVED and the origin of the event is CTI_VENDOR. The '*Expected Action*' for this event is, notify the Fusion application about the details of this inbound call. The newCommEvent API can be used to accomplish this and the 'UEF API to achieve this action' will be 'publish NewCommEvent' with caller details as payload. This is explained in the row (#1) in the below table.

Scenario 1 - Incoming Call to SERVICE_CENTER

An incoming call is initiated at the CTI_VENDOR and the event is listened at the PARTNER application. It is accepted at SERVICE_CENTER. The call is ended either at SERVICE_CENTER or at CTI_VENDOR.

Given below is the incoming call lifecycle functional diagram



Given below is the interaction diagram for the Incoming call scenario

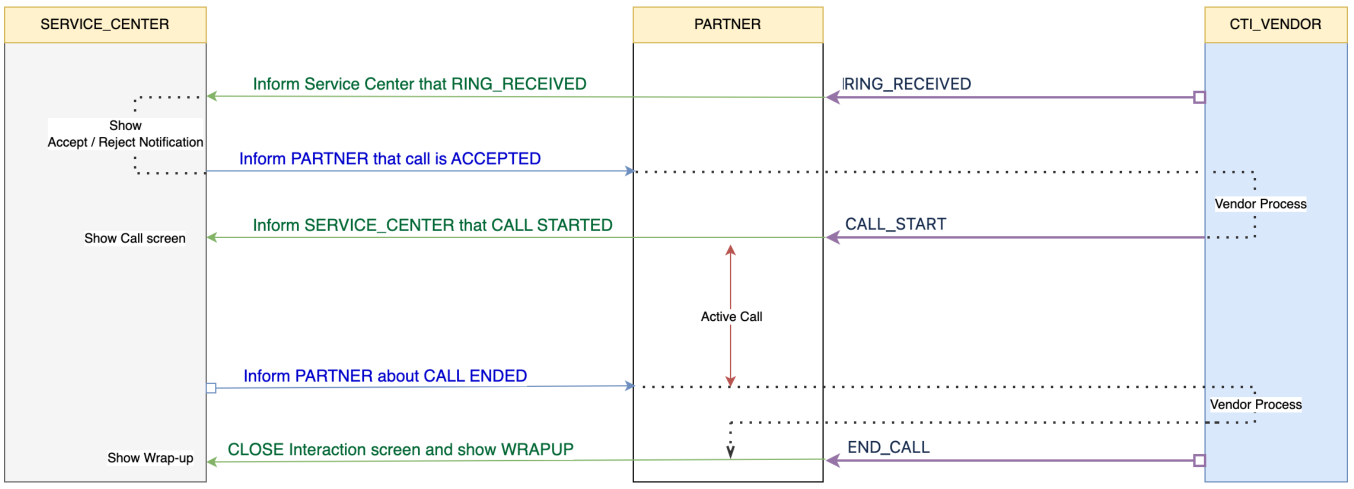

SERVICE_CENTER
PARTNER
CTI_VENDOR
Inform Service Center that RING_RECEIVED
RING_RECEIVED
Show Accept / Reject Notification
Inform PARTNER that call is ACCEPTED
Vendor Process
Inform SERVICE_CENTER that CALL STARTED
CALL_START
Show Call screen
Active Call
Inform PARTNER about CALL ENDED
Vendor Process
CLOSE Interaction screen and show WRAPUP
END_CALL
Show Wrap-up

Table: **CTI Generic Events and Actions Mapping Table**

#	Generic Event	Event Origin	Event Description	Expected Action	UEF API to achieve Expected Action
1	RING_RECEIVED	CTI_VENDOR	An inbound call	Show a notification in SERVICE_CENTER with Accept / Reject Button	PARTNER can publish a 'NewCommEvent' action to the SERVICE_CENTER with details like phone number.

NewCommEvent (fired from PARTNER to SERVICE_CENTER)

```
// Set up the request object
const request: IMcaNewCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('newCommEvent') as IMcaNewCommEventActionRequest;
request.getInData().setInDataValueByAttribute('SVCMCA_ANI', '9790846755');
request.getInData().setInDataValueByAttribute
('SVCMCA_COMMUNICATION_DIRECTION','ORA_SVC_INBOUND');
request.setEventId(eventId);
request.setAppClassification('ORA_SERVICE');
// Perform publish in phone context
const operationResponse: IOperationResponse = await phoneContext.publish(request);
```

2	ACCEPTED	SERVICE_CENTER	Agent Accepts the call by clicking the Accept button in the notification popup in SERVICE_CENTER	The PARTNER gets notified about this Accept event triggered at the SERVICE_CENTER and then PARTNER informs the CTI_VENDOR.	PARTNER can register a subscription for the UEF event 'onToolbarInteractionCommand' with a callback during initialization of PARTNER application. Whenever the agent clicks on Accept or Reject button, the callback registered with the above subscription is executed. From this callback, customer can call the CTI_VENDOR specific APIs to inform them about the agent action.

onToolbarIneractionCommand

```
const request: any = frameworkProvider.requestHelper.createSubscriptionRequest
('onToolbarInteractionCommand');
// Register a subscription with the service center
phoneContext.subscribe(requestForOutGoing, function(eventResponse){
    // eventResponse has the information about the agent's action in the service center
        switch(command) {
        case 'Accept':
        case 'Reject':
                case 'Disconnect':
    }
});
```

	REJECTED	SERVICE_CENTER	Agent rejects the call at the SERVICE_CENTER	The PARTNER gets notified about this Accept event triggered at the SERVICE_CENTER and then PARTNER informs the CTI_VENDOR.	'onToolbarInteractionCommand' is fired and the PARTNER get notified. The PARTNER can publish a CloseCommEvent action to SERVICE_CENTER with reason 'REJECT'.

CloseCommEvent - REJECT

```
const request: IMcaCloseCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('closeCommEvent') as IMcaCloseCommEventActionRequest;
request.setReason('REJECT');
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse = phoneContext.publish(request);
```

3	CALL_START	CTI_VENDOR	This is when both parties enter into active call mode.	Both SERVICE_CENTER and CTI can render suitable screen for an active call with caller details, End Call button, call time counter, etc.	Customer can publish StartCommEvent action to SERVICE_CENTER to inform the SERVICE_CENTER that the call is active.

StartCommEvent

```
const request: IMcaStartCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('startCommEvent') as IMcaStartCommEventActionRequest;
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse: IOperationResponse = await phoneContext.publish(request);
```

4	END_CALL	SERVICE_CENTER / CTI_VENDOR	An active call can be end from SERVICE_CENTER or from the Customer side (CTI_VENDOR)	The web component (that has call time and end call button) used during an active call in the SERVICE_CENTER is closed. If required, a call wrap-up element can be rendered.	If the EndCall is pressed at the SERVICE_CENTER side, 'onToolbarInteractionCommand' event is fired from SERVICE_CENTER with 'End Call' as event payload. This event is subscribed by the PARTNER and hence the callback registered with the subscription is executed. PARTNER can then inform the CTI_VENDOR about this event in the callback. PARTNER can publish CloseCommEvent action with proper reason like 'WRAPUP' to windup the active call and show the wrap-up window.

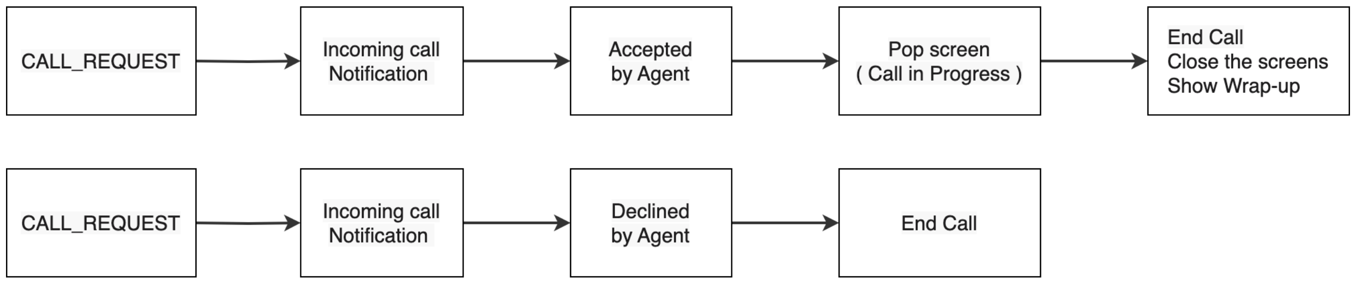
CloseCommEvent - WRAPUP

```
const request: IMcaCloseCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('closeCommEvent') as IMcaCloseCommEventActionRequest;
request.setReason('WRAPUP');
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse = phoneContext.publish(request);
```

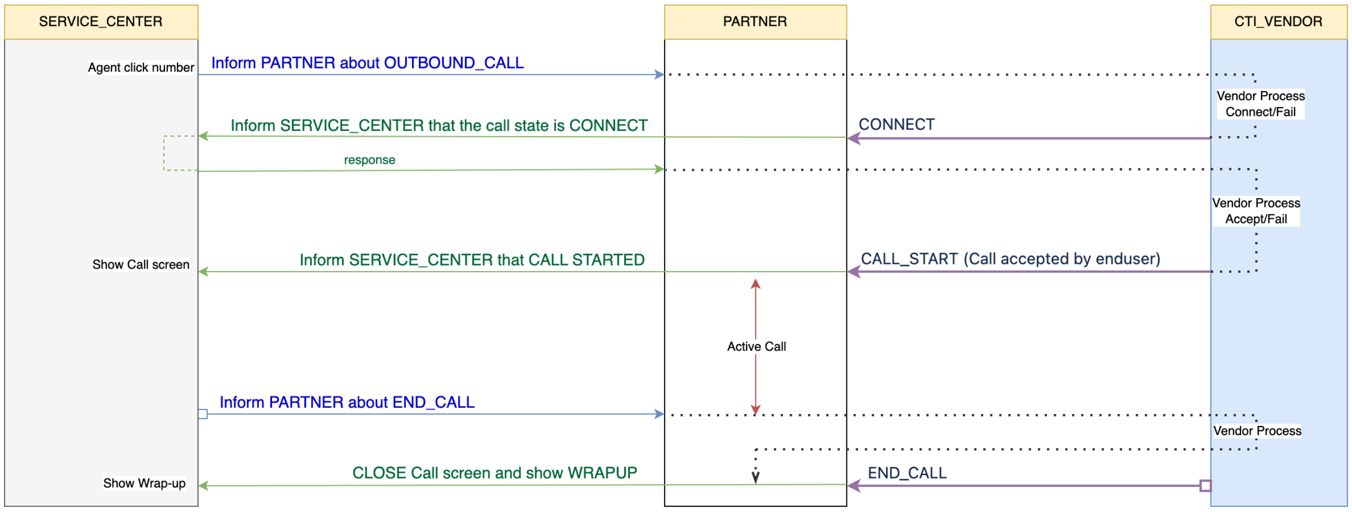
Scenario 2 - Outgoing Call from SERVICE_CENTER

Given below is the outgoing call lifecycle functional diagram

INBOUND call life cycle


CALL_REQUEST
Incoming call Notification
Accepted by Agent
Pop screen (Call in Progress)
End Call Close the screens Show Wrap-up
CALL_REQUEST
Incoming call Notification
Declined by Agent
End Call

Given below is the interaction diagram for a generic outgoing call scenario


SERVICE_CENTER
PARTNER
CTI_VENDOR
Agent click number
Inform PARTNER about OUTBOUND_CALL
Vendor Process Connect/Fail
Inform SERVICE_CENTER that the call state is CONNECT
CONNECT
response
Vendor Process Accept/Fail
Show Call screen
Inform SERVICE_CENTER that CALL STARTED
CALL_START (Call accepted by enduser)
Active Call
Inform PARTNER about END_CALL
Vendor Process
Show Wrap-up
CLOSE Call screen and show WRAPUP
END_CALL

#	Generic Event	Event Origin	Description	Expected Action	UEF API to achieve Expected Action
1	OUTBOUND_CALL	SERVICE_CENTER	An agent click on a number in the SERVICE_CENTER to make an outbound call	PARTNER is notified about this and it informs the CTI_VENDOR	PARTNER can register an onOutgoingEvent subscription with UEF during initialization. When an outGoingEvent is occurred, the callback registered executed. The PARTNER can inform CTI_VENDOR about this action in callback.

onOutgoingEvent

```
const request: any = frameworkProvider.requestHelper.createSubscriptionRequest
('onOutgoingEvent');
request.setAppClassification('ORA_SERVICE');
phoneContext.subscribe(request, (eventResponse: IEventResponse) => {
    // Inform CTI about the outbound call
});
```

#	Generic Event	Event Origin	Description	Expected Action	UEF API to achieve Expected Action
2	CONNECTED	CTI_VENDOR	The end user is available but not accepted the call yet.	A pop up can be shown at the end user with Accept / Reject buttons.	PARTNER can publish NewCommEvent action to SERVICE_CENTER using phone context.

NewCommEvent (fired from PARTNER to SERVICE_CENTER)

```
// Set up the request object
const request: IMcaNewCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('newCommEvent') as IMcaNewCommEventActionRequest;
request.getInData().setInDataValueByAttribute(key, value);
request.setEventId(eventId);
request.setAppClassification('ORA_SERVICE');
// Perform publish in phone context
const operationResponse: IOperationResponse = await phoneContext.publish(request);
```

#	Generic Event	Event Origin	Description	Expected Action	UEF API to achieve Expected Action
	FAILED	CTI_VENDOR	The end user may be offline or number is invalid	The PARTNER can inform the service center that the end user is not available.	PARTNER can publish a outBoundCommError followed by CloseCommEvent with reason - 'MISSED'.

OutboundCommError

```
const request = frameworkProvider.requestHelper.createPublishRequest
('outboundCommError');
request.setCommUuid(EventId);
request.setErrorCode(errorCode);
request.setErrorMsg(errorMessage);
const operationResponse: IOperationResponse = await phoneContext.publish(request);
```

CloseCommEvent - MISSED

```
const request: IMcaCloseCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('closeCommEvent') as IMcaCloseCommEventActionRequest;
request.setReason('MISSED');
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse = phoneContext.publish(request);
```

3	ACCEPTED	CTI_VENDOR	The end user clicked on the Accept button in the pop up and the active call starts.	SERVICE_CENTER and CTI displays suitable screen for an active call with caller details, End Call button, call time counter, etc.	Publish StartCommEvent action from CTI to SERVICE_CENTER with data we get from the call connection response.

StartCommEvent

```
const request: IMcaStartCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('startCommEvent') as IMcaStartCommEventActionRequest;
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse: IOperationResponse = await phoneContext.publish(request);
```

	REJECTED	CTI_VENDOR	The end user clicked on the reject button	The PARTNER can inform the service center that the call has been rejected	If the end user has rejected the call, the CTI response will be REJECTED. The customer can publish a CloseCommEvent action to SERVICE_CENTER with reason 'REJECT'.

CloseCommEvent - REJECT

```
const request: IMcaCloseCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('closeCommEvent') as IMcaCloseCommEventActionRequest;
request.setReason('REJECT');
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse = phoneContext.publish(request);
```

4	END_CALL	SERVICE_CENTER / CTI_VENDOR	An active call can be end from SERVICE_CENTER or from CTI_VENDOR.	Active call element in the SERVICE_CENTER exits and if required a call wrap-up element can be rendered	If EndCall is pressed from SERVICE_CENTER, 'onToolbarInteractionCommand' event is fired from SERVICE_CENTER with 'End Call' as event payload. The customer can inform the CTI_VENDOR about the end call Customer can Publish CloseCommEvent with proper reason like 'WRAPUP'.

CloseCommEvent - WRAPUP

```
const request: IMcaCloseCommEventActionRequest = frameworkProvider.requestHelper.
createPublishRequest('closeCommEvent') as IMcaCloseCommEventActionRequest;
request.setReason('WRAPUP');
request.getInData().setInDataValueByAttribute(property, value);
request.setEventId(EventId);
request.setAppClassification('ORA_SERVICE');
const operationResponse = phoneContext.publish(request);
```