ORACLE

Integrating Siebel CRM with Microsoft Teams for Real-Time Notifications and Improved Collaboration – Recipe

April, 2025, Version [1.0]

Public

ORACLE

Purpose statement

This document has been created by the Siebel Center of Excellence team for informational purposes only. It aims to provide valuable insights and guidance on the subject matter while serving as a reference for stakeholders. This document does not constitute a binding agreement or official policy.

Disclaimer

ORACLE

Table of contents

ORACLE

Summary

This whitepaper outlines the integration of Siebel CRM with Microsoft Teams to enhance collaboration and streamline notification workflows. The integration facilitates seamless real-time communication by enabling MS Teams to notify users of significant changes made within the Siebel CRM system. It leverages RESTful APIs, adaptive cards, and dynamic deep linking, ensuring that all Siebel updates are communicated to relevant stakeholders directly in MS Teams.

Introduction

Siebel CRM is a robust customer relationship management platform that handles a wide variety of business processes. Integrating Siebel with Microsoft Teams enables users to remain updated about critical business events without having to switch between applications.

The purpose of this integration is to enhance productivity, improve communication, and reduce operational inefficiencies by pushing Siebel CRM updates to Microsoft Teams in a structured and actionable format.

Objectives of the Integration

The key objectives of this integration include:

- Real-time notifications to users in MS Teams upon key events within Siebel.
- Ensuring dynamic deep linking to specific records in Siebel from MS Teams, allowing users to access critical data quickly.
- Streamlining communication and collaboration within MS Teams by providing users with updates formatted using adaptive cards.
- Offering the flexibility to notify the correct set of users based on the changes made in Siebel.

Pre-requisites

For Development

- MS Teams Toolkit Extension should be installed in Visual Studio Code
- Microsoft 365 Developer account with Custom app upload permission enabled (For Testing the Integration with Microsoft Teams)
- Azure Entra ID account with access to Azure services (To create the Bot using Microsoft Bot Framework)

For Usage

- The user should be using the same email to login to both Siebel and MS Teams

Functionality

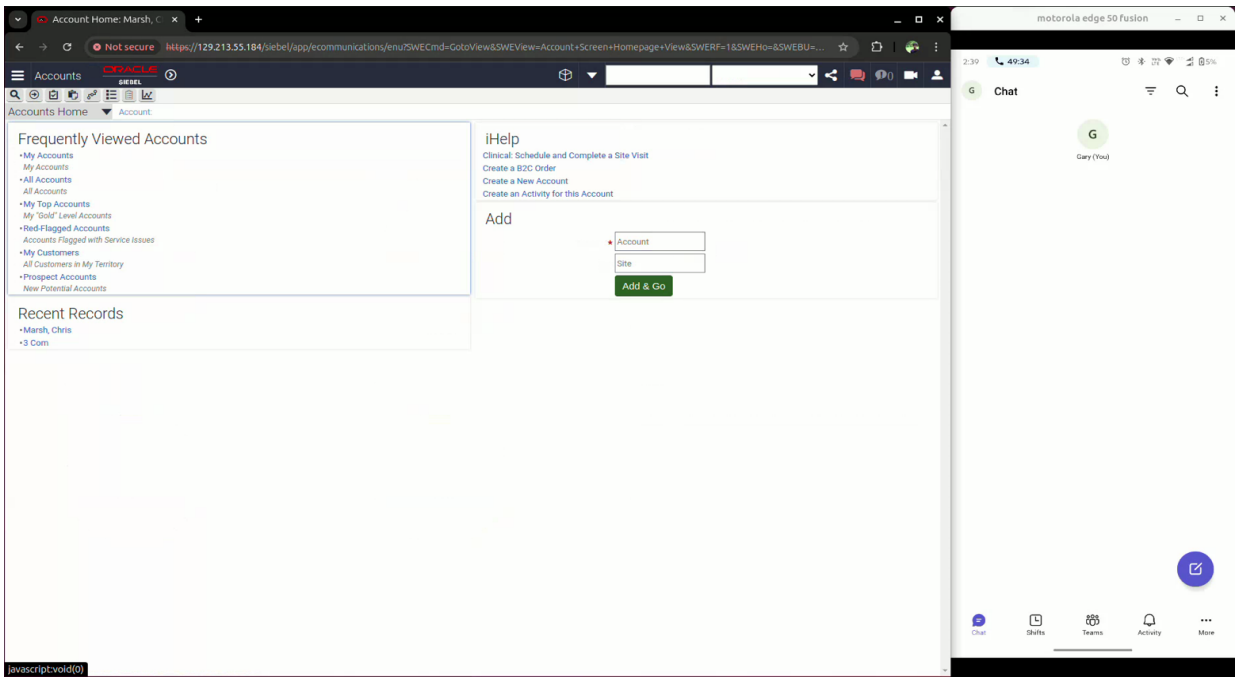
Flow 1

Siebel User (Field Service Agent) receives Assigned Appointment details (as a chat message) in their MS Teams application

Steps

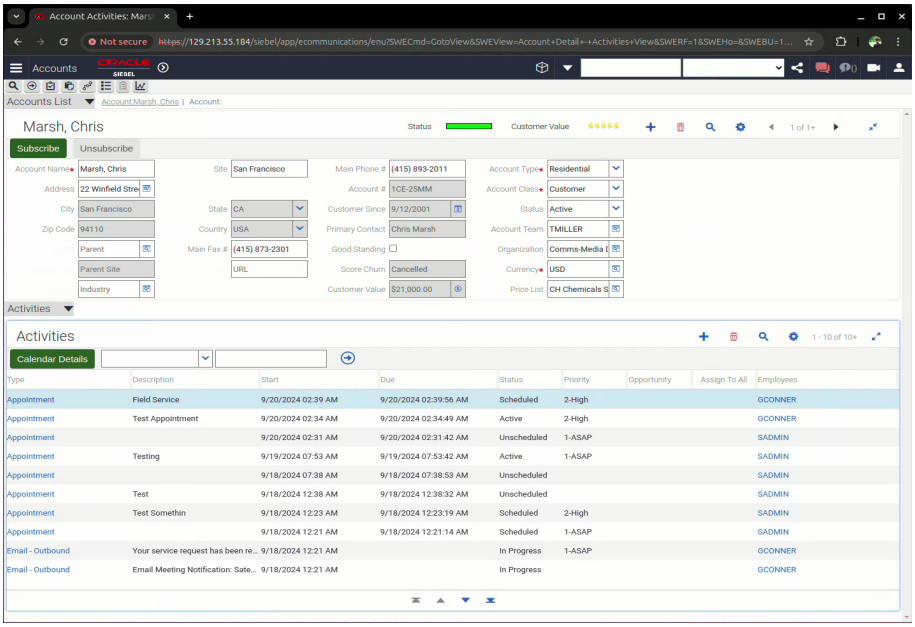
1. Go To Siebel ecommunications UI(Login as any admin user) and go to Accounts Page. Install MS Teams on a mobile device and be logged in to MS Teams as the Field Service Agent.

Figure 1 – Siebel View Logged as Service Administrator & MS Teams App logged in as Field Service Agent



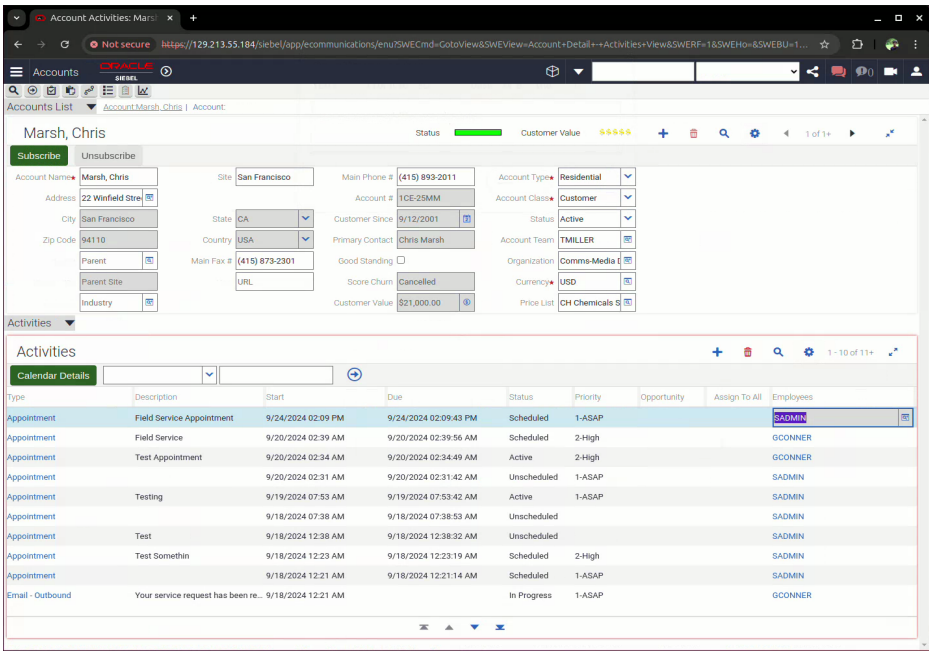
2. Choose the account you want to create the Activity for. From the second applet selector (Drop-down list), choose Activities

Figure 2 - Siebel Account Details Page



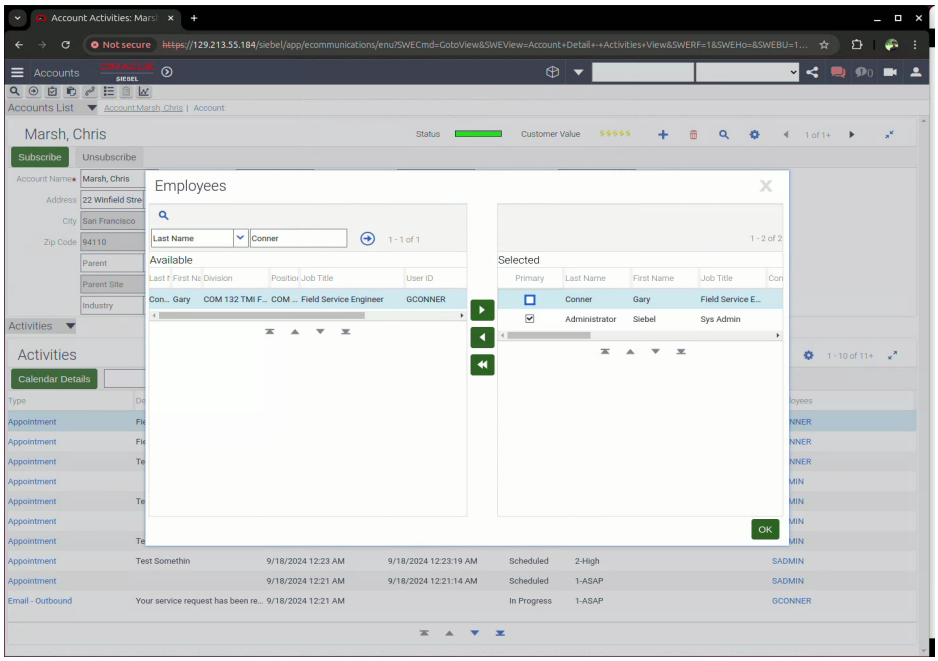
3. Create a new Activity for the Account and add 'Description', 'Status' and 'Priority' for the Activity

Figure 3 - Activity created for the account Marsh, Chris



4. Click on the Employee column. From the MVG applet, choose the user corresponding to the MS Teams user and set them as the primary Employee and finally Click on 'Ok'.

Figure 2 - Assign the Activity to Field Service Agent



5. Save the record. After a few seconds, you will be notified about the Activity on the MS Teams application. You can click on the deep link attached in the notification message and see the Siebel UI as well

Confidential - Oracle Restricted

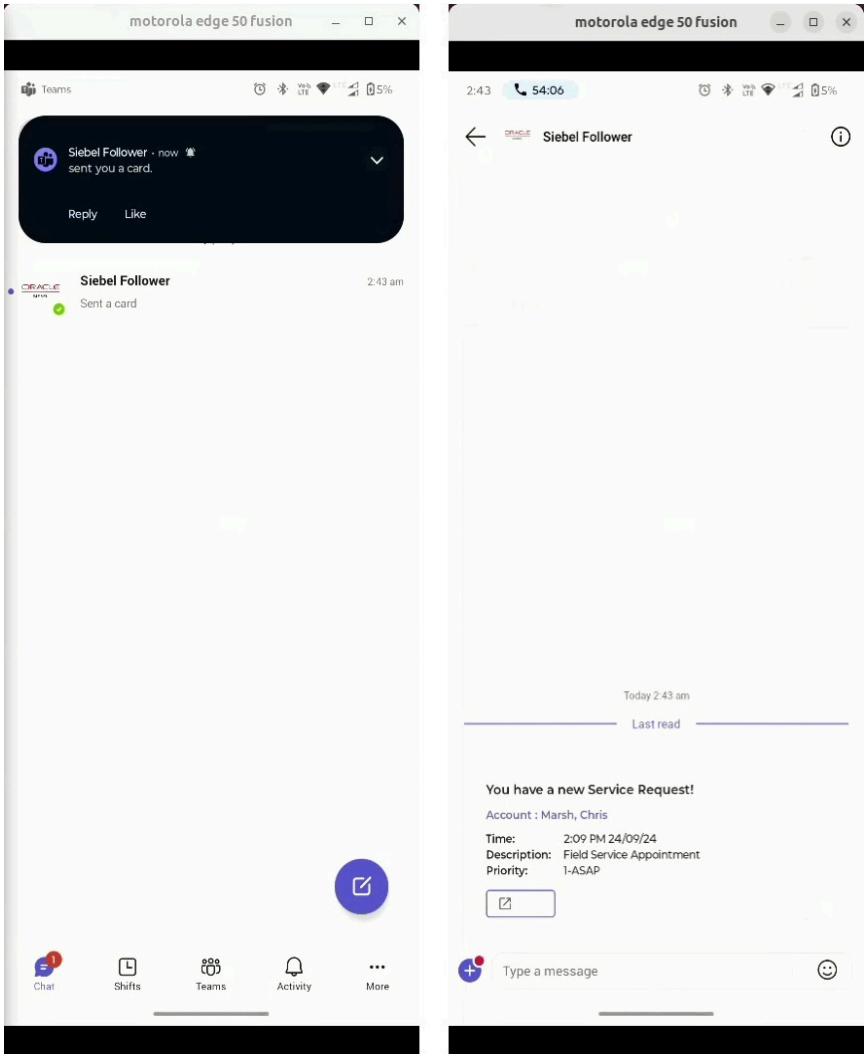


Figure 3 - MS Teams Notification for the Assigned Field Service Agent

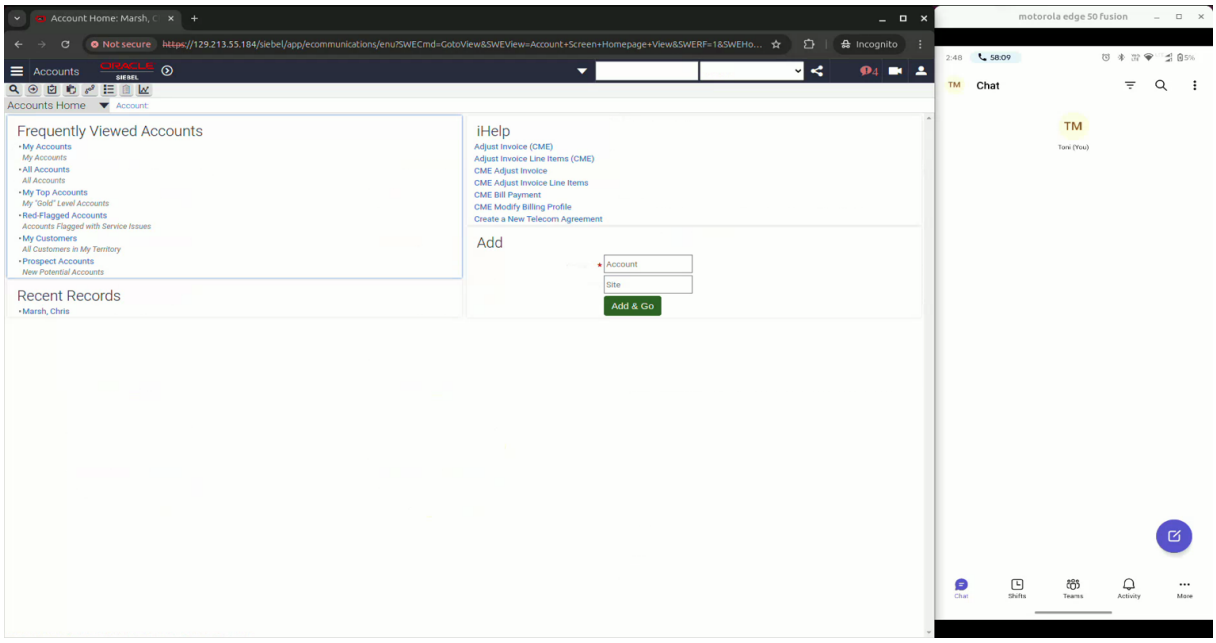
Flow 2

Siebel user clicks on the 'Follow' button on a particular object (Business component Record) in Siebel UI. The user then gets a notification for each activity (Modification) performed on that record.

Steps

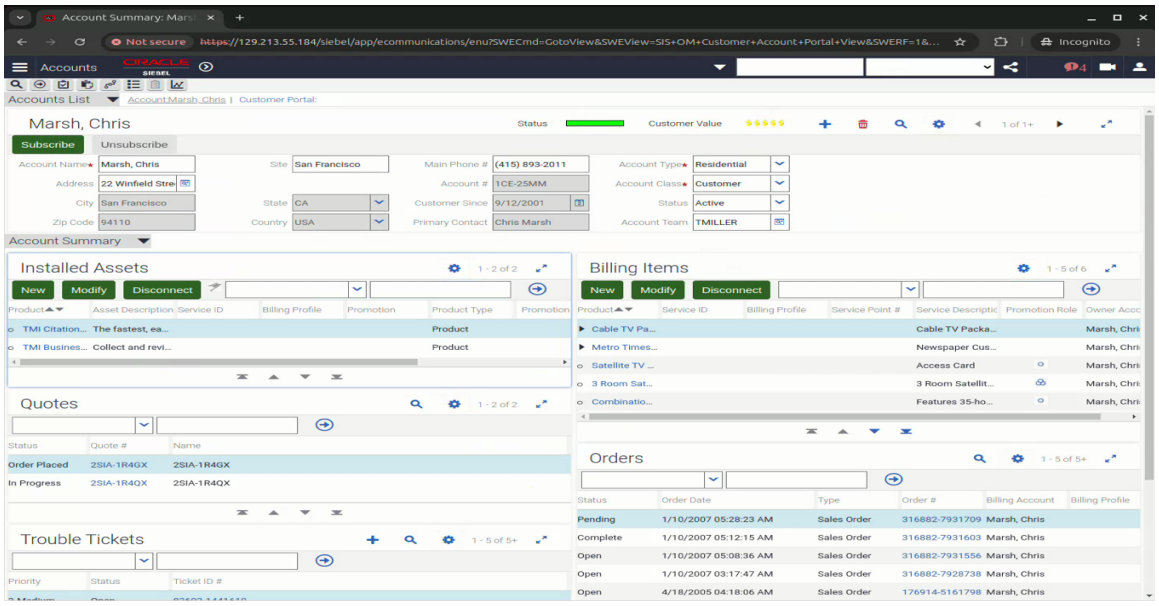
1. Go To Siebel ecommunications UI (Login as a user who can view account details) and go to Accounts Page. Install MS Teams on a mobile device and be logged in to MS Teams as the same user (Tony Miller/ TMILLER in this example)

Figure 6 - Siebel& MS Teams Views Logged in as Customer Service Agent



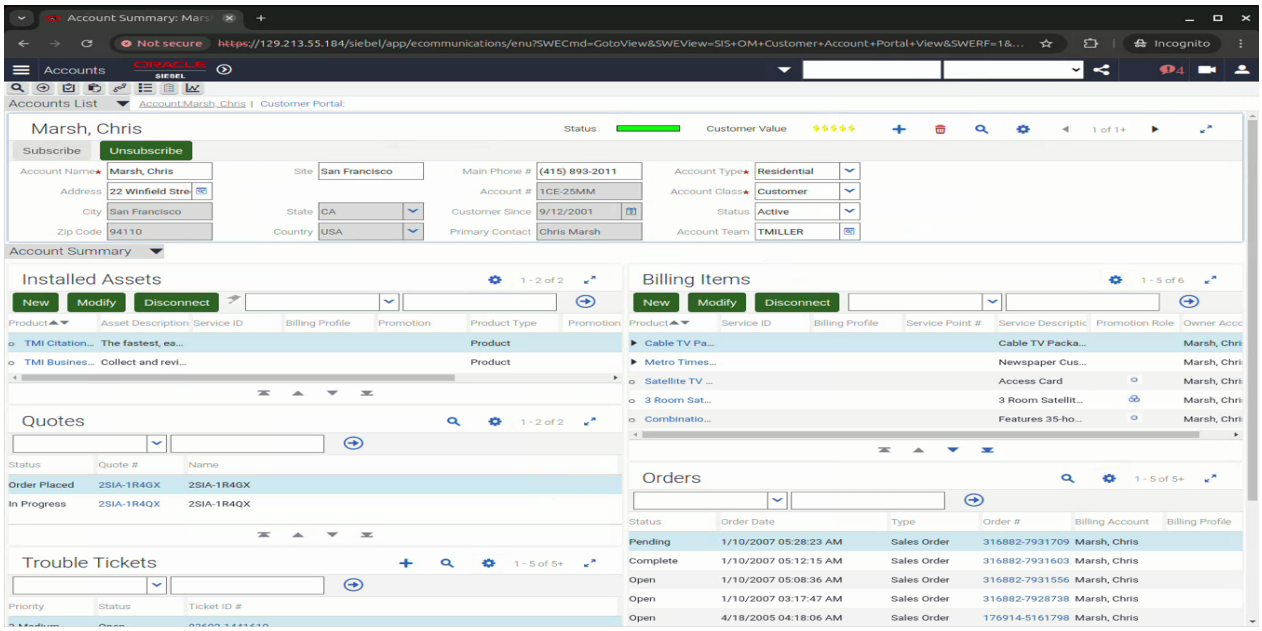
2. Go to the account that you want to 'Follow'

Figure 7 – Siebel Account Details Page



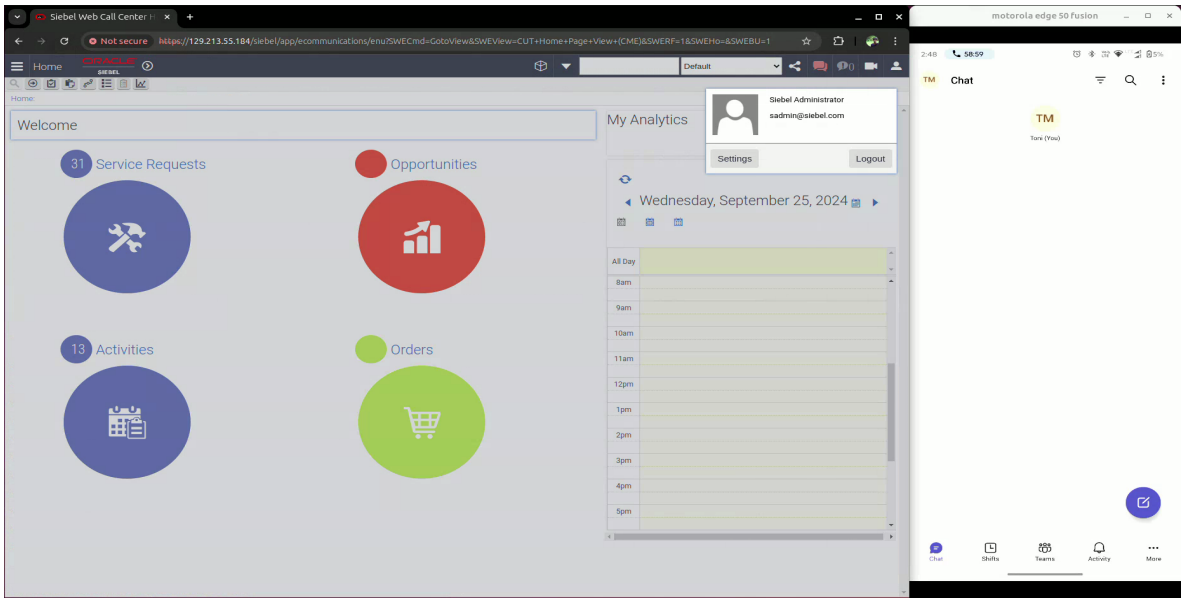
3. Click on the Subscribe Button

Figure 8 - Siebel Account Details Page - Account Subscribed



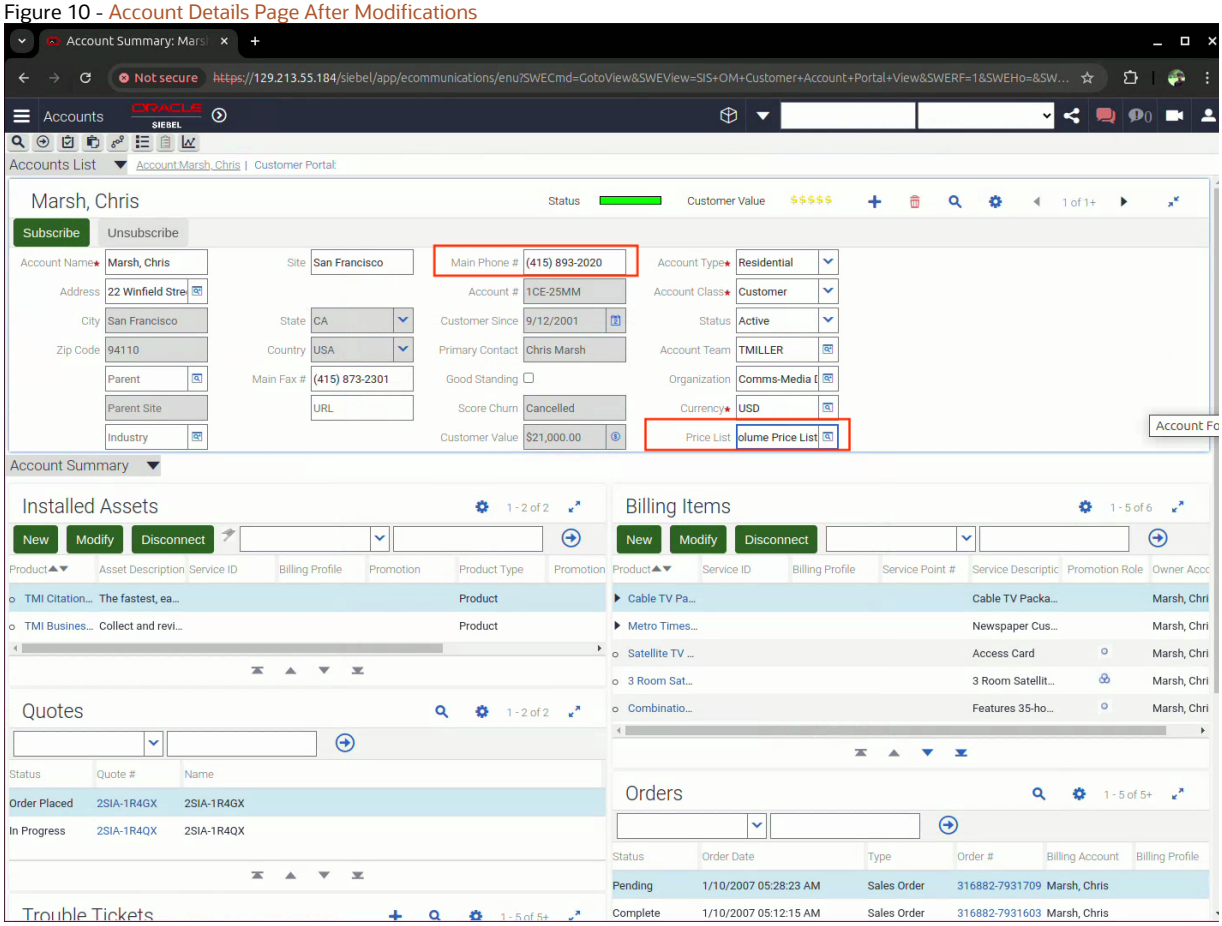
4. Log out from Siebel and login as a user who can modify Account Details (SADMIN in this example)

Figure 4 - Siebel View Logged in as Administrator & MS Teams Logged in as Customer Service Agent



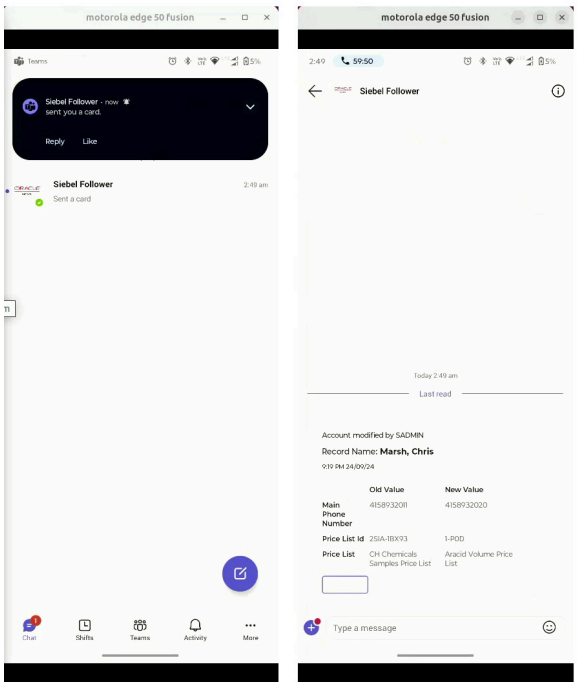
5. Go To Accounts Page and choose the Account that the initial user chose to 'Follow'. Modify the values of the field values from the Account Details section. eg: 'Phone Number', 'Price List'

Figure 10 - Account Details Page After Modifications



6. Save the Account after making these changes. After a few seconds, you will be notified about the Modifications performed on the Account, in your MS Teams application. You can click on the deep link attached in the notification message and see the Siebel UI of the Account as well

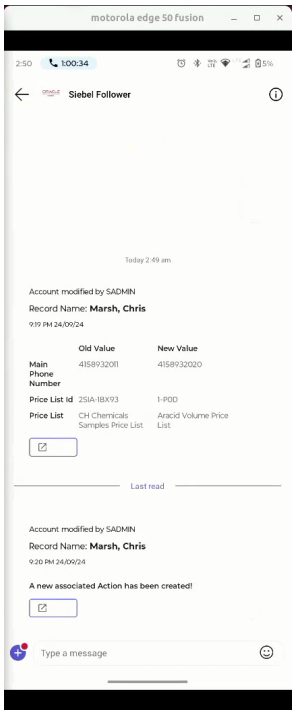
Figure 11 - MS Teams Notification for the Customer Service Agent (Agents Following the Account)



7. In Siebel, select the 'Activity' applet from the Drop-down list and create a new Activity for the Account.

8. After Saving the Activity, you will be notified about the new Activity created for the Account, on your MS Teams application. You can click on the deep link attached in the notification message and see the Siebel UI of the Account, where you can view the Activity details as well

Figure 5 - MS Teams Notification for the Customer Service Agent about a new Activity Created



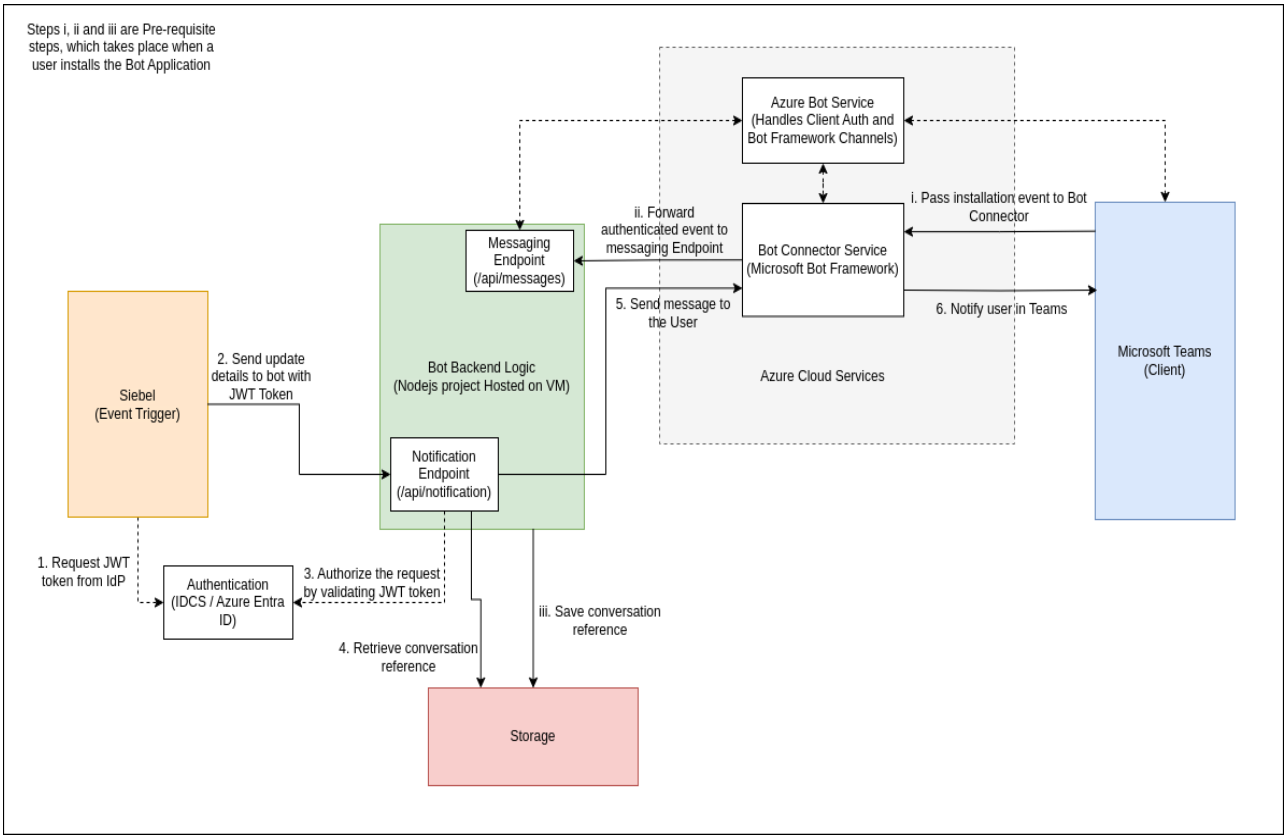
Confidential - Oracle Restricted

Technical Design

MS Teams Custom Bot

MS Teams Custom Bot design will be common for both the User Flows

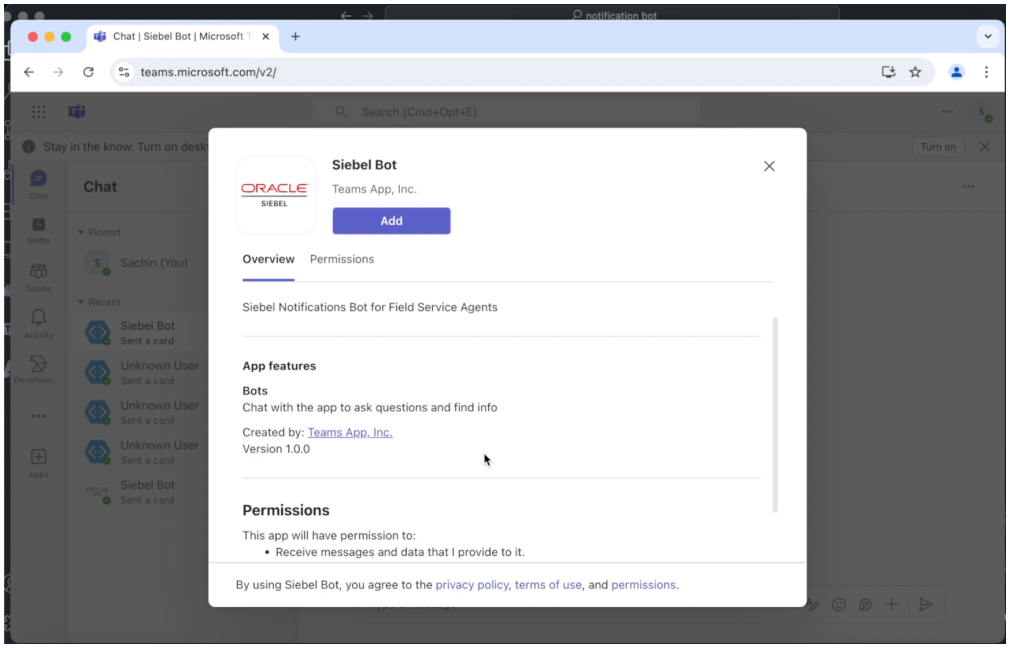
Figure 6 - MS Teams Custom Bot Technical Design



Pre-Requisite Steps

1. The user installs the custom bot app in their Microsoft Teams environment. Teams Client creates an event and passes it to the Messaging endpoint(/api/messages), through Bot Connector Service

Figure 14 - User Adds the Bot on their MS Teams Application



2. Event arrives at the messaging endpoint as HTTP Request. Bot Framework SDK validates the Authentication Token and the payload in the request
3. Bot Framework SDK processes the Conversation Reference obtained from the event (which includes user details like the user ID) and saves it in storage according to the specifications done in the Backend Logic. As our bot is a Proactive bot, we need to save Conversation Reference to initiate the chat with a user. Bot-Framework-SDK provides in-built methods to find a particular user's conversation reference using their email-id.

Bot Backend Flow

1. On Event trigger, Siebel requests a JWT token from Azure AD by using the client ID and client secret obtained from the respective identity provider (IdP). This token is necessary to access the protected backend logic of the custom bot. (Client Id and Client secret could be stored in Siebel as environment variables and be accessed through code.
2. On obtaining the JWT Token, Siebel sends a POST request to the bot's protected notification endpoint(/api/notifications). This request includes the activity details and the email ID of the assigned user
3. The bot backend logic checks the validity of the JWT token to authorize the Siebel client to access the protected Notification API. If the token is valid, bot proceeds with the request.
4. Bot retrieves the conversation reference stored during the bot installation steps (MS Teams Custom Bot Flow steps 1-3) using the user ID obtained from the POST payload
5. If the user is found in the conversation reference storage, the bot uses the sendAdaptiveCard method to send a message to the user. The bot formulates the reply message using Adaptive Cards, ensuring the message is tailored to the activity details provided by Siebel.
6. Teams Client Obtains the message from the bot and sends notification to the User
7. If the user is not found (i.e., they have not installed the bot), the bot returns an error message, indicating that the user needs to install the bot app in Microsoft Teams

Confidential - Oracle Restricted

Siebel Design Flows

Flow - 1

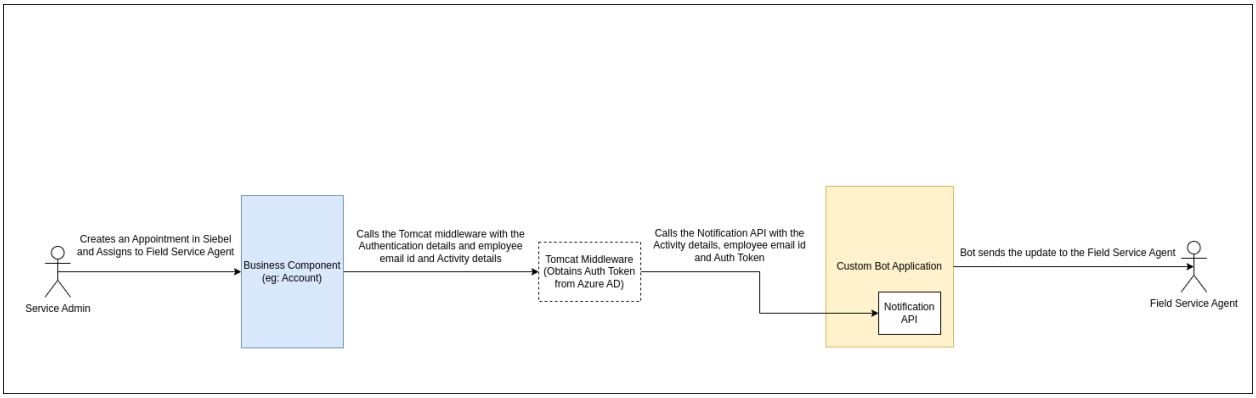


Figure 7 - Field Service Agent Use Case - Functional Design

1. 'Service Admin' Creates a new Activity and assigns it to a 'Field Service Agent' in Siebel
2. Upon activity creation, Siebel requests a JWT token from Azure AD by using the client ID and client secret obtained from the respective identity provider (IdP) through a tomcat middleware service
3. On obtaining the JWT Token, Siebel(tomcat middleware) sends a POST request to the bot's protected notification endpoint(/api/notifications). This request includes the activity details and the email ID of the assigned user
4. The bot backend logic checks the validity of the JWT token to authorize the Siebel client to access the protected Notification API. If the token is valid, bot retrieves the user's conversation reference and sends a message to the user by using Adaptive Cards. This message will be tailored to the activity details provided by Siebel

Flow - 2

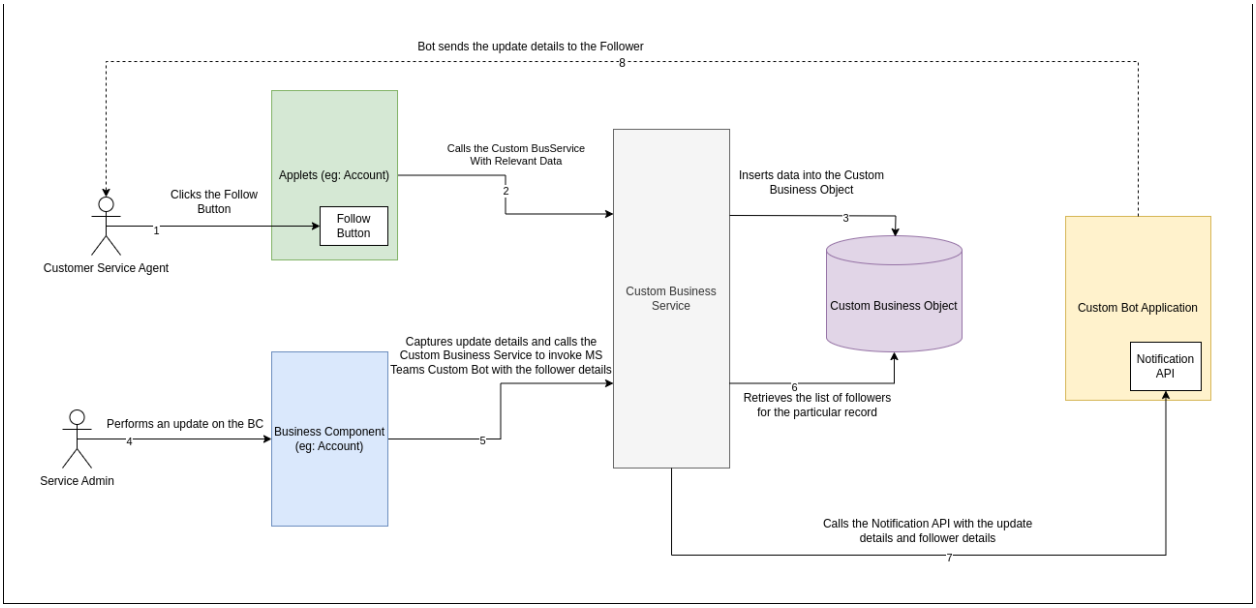


Figure 8 - Customer Service Agent Use Case - Functional Design

1. '' Clicks on the 'Follow' button on a Business Component record from the Siebel UI(Applet UI)
2. Siebel calls a custom Business Service which interacts with the Custom Business Object to handle the storage
3. User Id, Record Id, Business Component name and Deep link gets stored in a Business Object

4. Whenever a user makes a change on the Followed record, step 5 begins
5. Siebel Business Component Event handlers invoke the Custom Business Service methods to fetch the list of users who are following the record and to invoke Notification API
6. Custom Business Service queries the Business Object for Users following the record
7. Siebel (Custom Business Service) sends a request to the bot's notification endpoint(/api/notifications). This request includes the modification details and the email ID of the followed user. MS Teams Custom Bot processes the request
8. Bot retrieves the conversation reference stored during the bot installation steps (step 1-3) using the user ID obtained from the POST payload
9. If the user is not found (i.e., they have not installed the bot), the bot returns an error message, indicating that the user needs to install the bot app in Microsoft Teams
10. If the user is found in the conversation reference storage, bot formulates the reply message using Adaptive Cards, ensuring the message is tailored to the update details provided by Siebel
11. Bot sends the update details to the user.

Implementation

MS Teams

We can leverage MS Teams Toolkit to setup a bot which can send notifications to users by using adaptive cards. Teams Toolkit provides boilerplate 'Proactive Notification Bot' code which we can enhance according to our scenario.

Deliverables

We have modified the boilerplate code MS Teams Toolkit provides for creating a Proactive Notification Bot to suit our use case.

FollowBot - compressed.zip

Steps

1. Extract and open the folder using Visual Studio Code.
2. Ensure npm version is 10.8.1 and node version is v20.16.0
3. Install npm packages using *npm install*
4. Run *npm run generate-guid* (generates a unique guid to create the required Azure resources easily through a script)
5. Go to the Teams Toolkit extension and Login into Microsoft 365 account. Run the 'local' script under 'Environment' section in Teams Toolkit extension

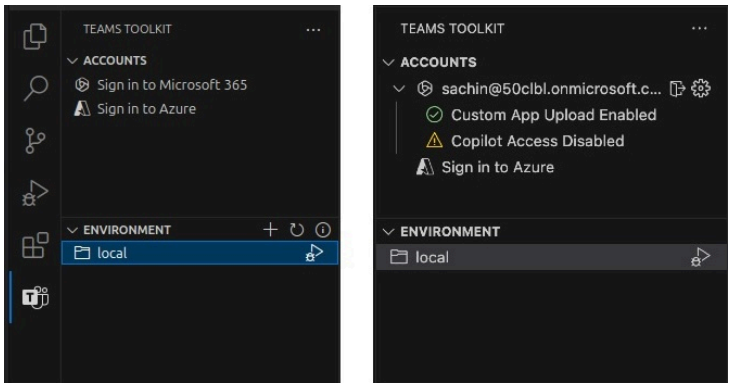


Figure 9 - VS Code Teams Toolkit Extension

6. Login into your teams account in the window that automatically pops up and Install the Bot (Refer: Figure 14)
7. Send a POST request to your Custom Bot for testing

```
{
    "user":"<MS Teams user id (email id)>",
    "type": "Activity",
    "accountName": "3 Com",
    "time": "07/26/2024 03:56:03",
    "url":"https://example.com/",
    "description":"Test Appointment",
    "priority":"1-ASAP",
    "deepLinkUrl": "https://<Siebel deep link url>"
}
```

The code base contains snippets to turn on the Authentication of Notifications API using Azure Entra ID Auth Token

You can run npm run dev:teamsfx after these steps to run the Custom Bot Application without opening a chrome window

Security

Notifications invocation Endpoint

We need to make sure that the Notification endpoint that we are exposing from the bot logic to accept Activity details payload allows only authorised access

For authenticating the calls from the Siebel application, we can use either Oracle IDCS/Azure Entra ID as the Identity Provider. We need to verify the jwt token inside the bot backend logic and authorise the access

We are exposing 'api/notification' API endpoint from the bot backend server.

Using Azure Entra ID is recommended for securing the Bot Backend logic, as there are npm packages recommended by Azure to verify the Bearer token provided by them.

Azure provide a 'BearerStrategy' by using 'passport' and 'passport-azure-ad' npm packages to verify the Authentication Token provided by them

Configuring a Client Application in Azure AD to access a Resource Application

1. Assuming that we already have an App Registration for our bot app(Automatically gets created after performing the steps in Deliverables section), this would be our Resource Application

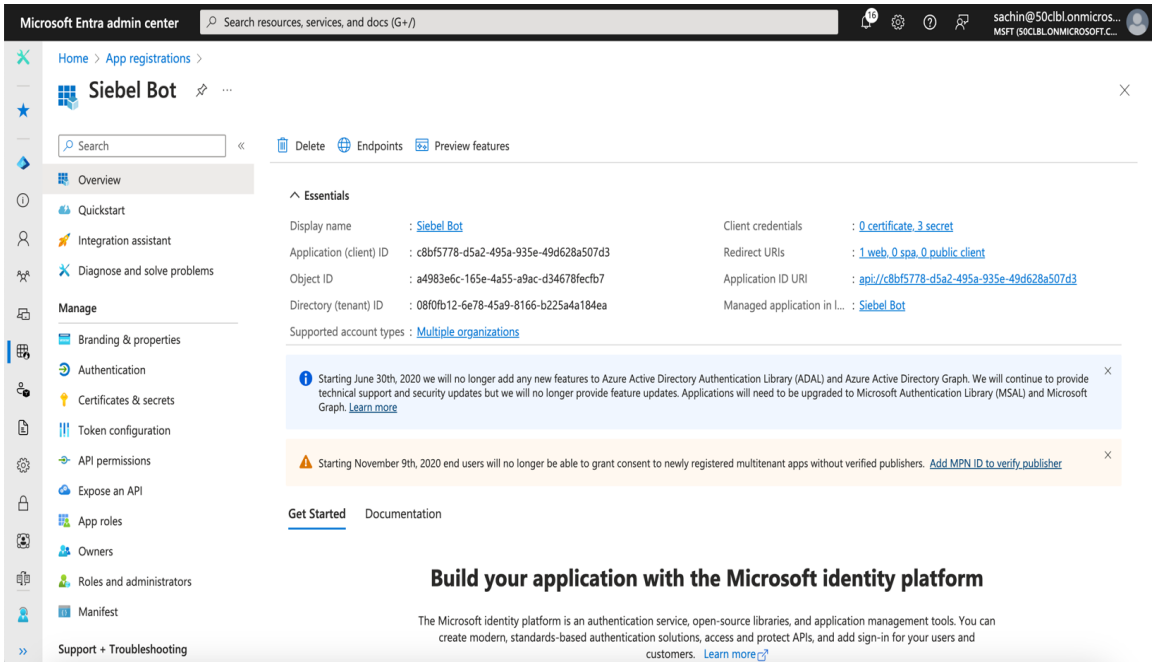
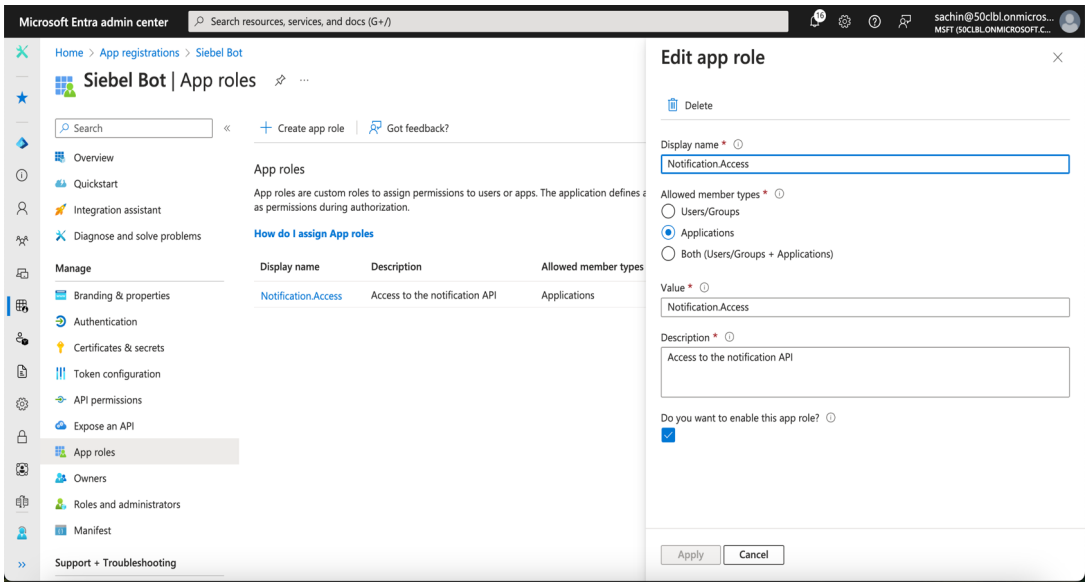


Figure 10 - Azure AD App Registration Details Page of the Resource Application (Custom Bot)

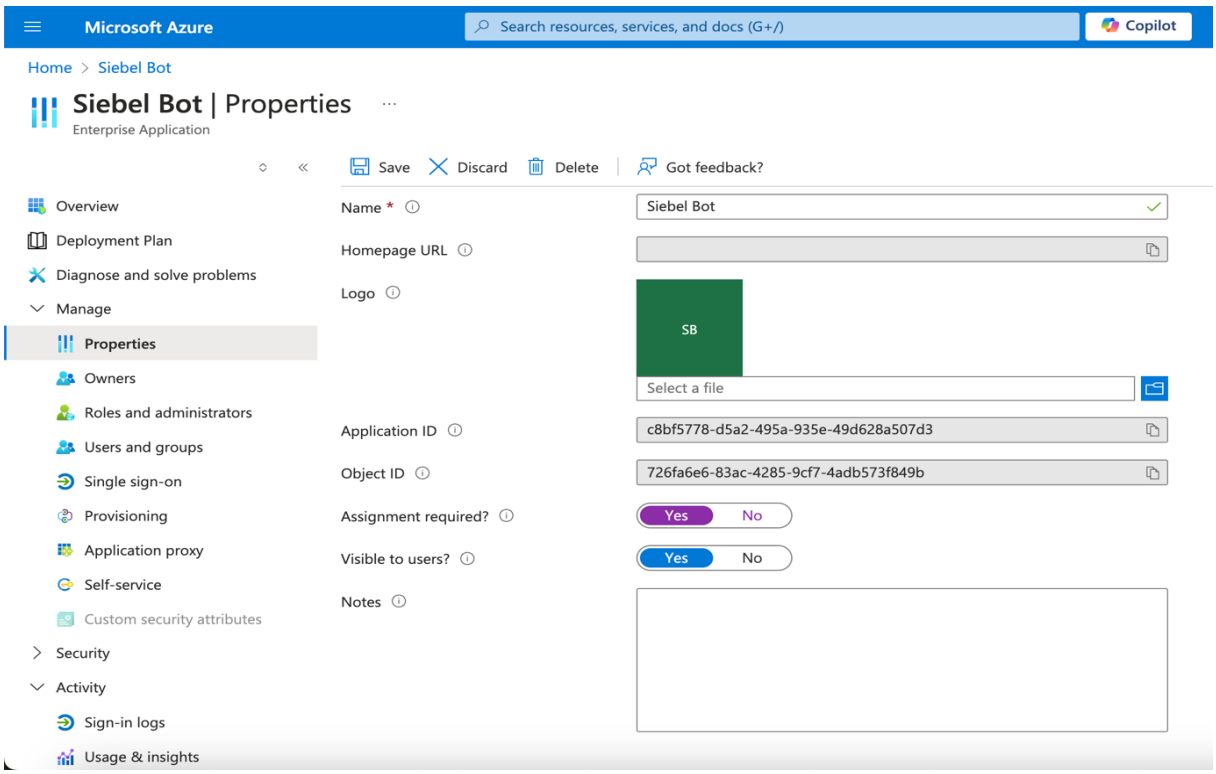
2. Click on Application URI in the App Roles tab for the above mentioned App Registration and add a new App Role as below

Figure 19 - Azure AD App Registrations - Add App Roles for the Resource Application



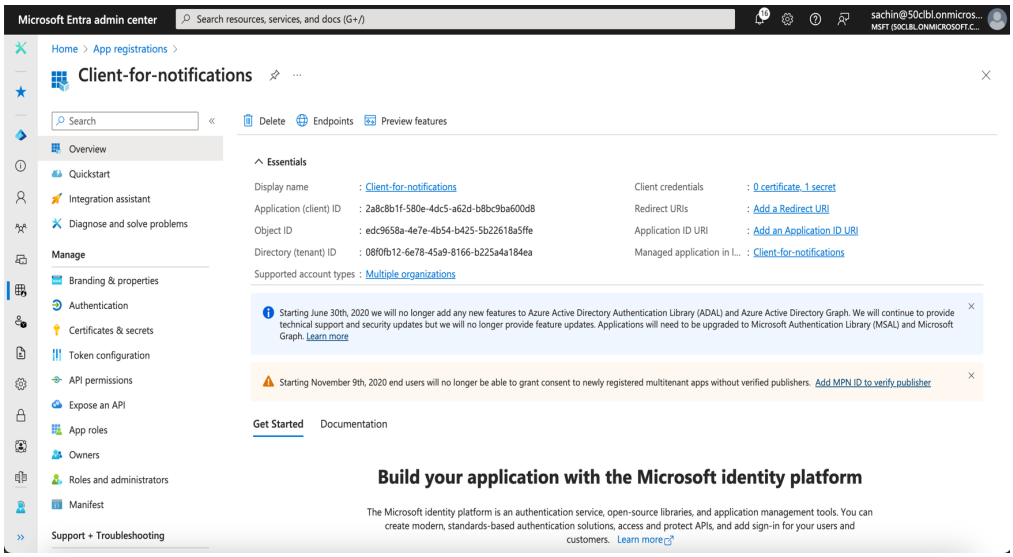
3. Go to Applications → Enterprise Application in Entra ID and choose the bot. In Manage → Properties tab, make sure 'Assignment required' is turned to 'Yes' - Unless this is turned to 'Yes', every app registration will be able to get an auth token for the Resource application (we'll have to rely on checking for the 'role' claim in the token)

Figure 11 - Azure AD Enterprise Application - Properties View for the Resource Application



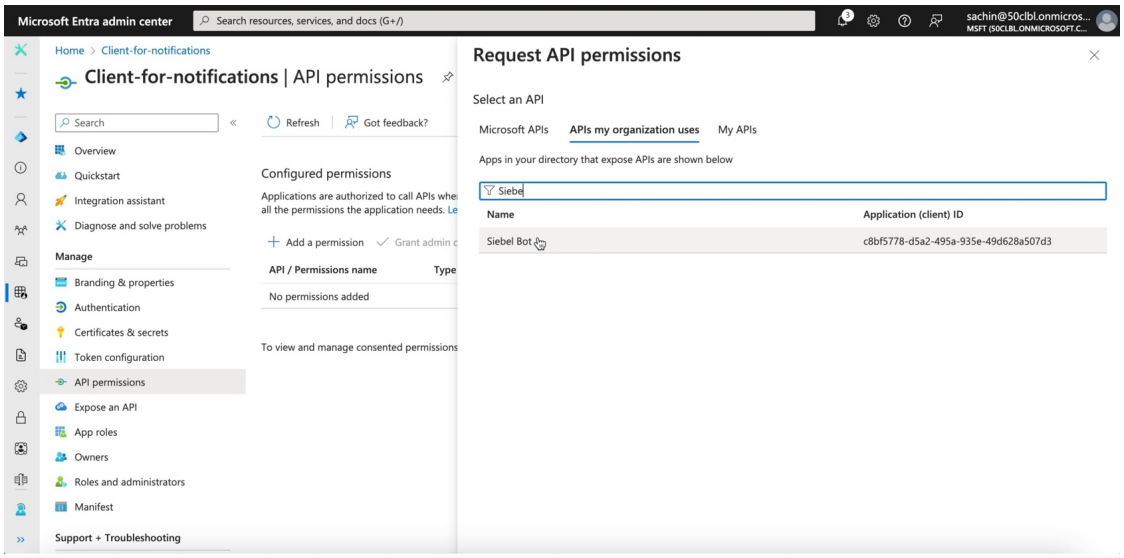
4. Create a new App Registration for a Client Application to access the Resource Application

Figure 21 - Azure AD App Registrations Page for the Client Application



5. Go to API permissions in the client Applications and add the App Role created in the second step. For this, click on ‘Add a Permission’, Choose ‘APIs my organization uses’, search for the ‘Siebel Bot’ Application and choose ‘Notification.Access’ from Application Permissions

Figure 22 - Azure AD - Add API Permissions for Client Application



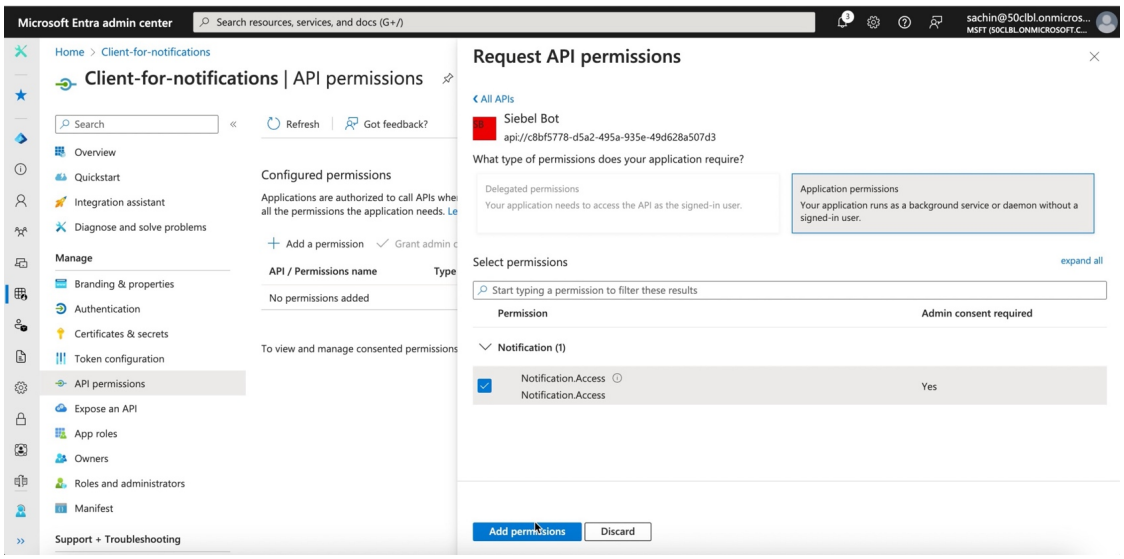
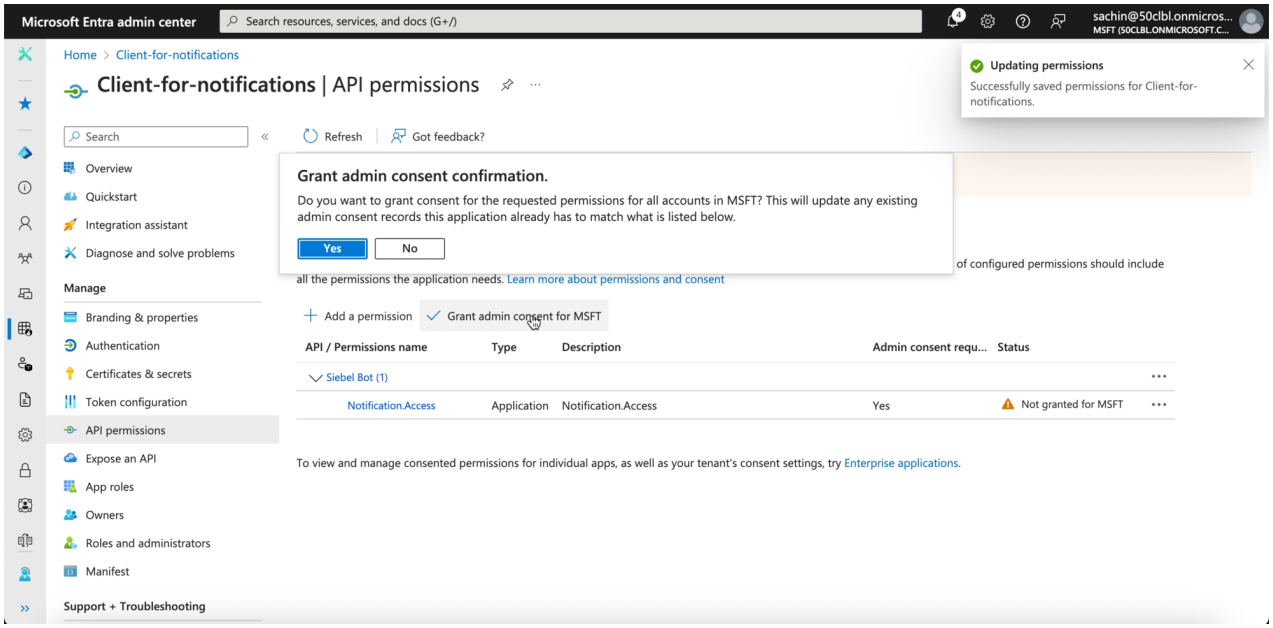


Figure 23 - Azure AD - Grant Admin consent for the added API Permissions

→



Obtaining token for client credentials flow

```
curl --location 'https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Basic <base64-encoded<client_id:client_secret>>' \
--data-urlencode 'scope=<scope-for-the-bot>.default'
--data-urlencode 'grant_type=client_credentials' \
```

Authorization of Bearer token from Azure using 'passport' and 'passport-azure-ad' packages in node-js

```
const passport = require('passport');// For Azure Auth for notification endpoint
const BearerStrategy = require('passport-azure-ad').BearerStrategy; // For Azure Auth for
notification endpoint
const { TeamsBot } = require("./teamsBot");

// For Azure Auth for notification endpoint
const options = {
  identityMetadata:
`https://login.microsoftonline.com/${process.env.TEAMS_APP_TENANT_ID}/${config.metadata.vers
ion}/${config.metadata.discovery}`,
  clientID: process.env.BOT_ID,
  audience: process.env.BOT_APP_URI ? process.env.BOT_APP_URI : "api://"+process.env.BOT_ID,
  validateIssuer: config.settings.validateIssuer,
  passReqToCallback: false,
  issuer: `https://sts.windows.net/${process.env.TEAMS_APP_TENANT_ID}/`
}

const bearerStrategy = new BearerStrategy(options, (token, done) => {
  done(null, { }, token);
}
);

const teamsBot = new TeamsBot();

const server = restify.createServer();
server.use(restify.plugins.bodyParser());


server.use(passport.initialize()); // for azure auth for notification endpoint
passport.use(bearerStrategy); // for azure auth for notification endpoint

// POST endpoint protected using Azure token validation
server.post(
  "/api/notification", passport.authenticate('oauth-bearer', {session: false}),
  restify.plugins.queryParser(),
  restify.plugins.bodyParser(), async (req, res) => {
    try {
//Place your logic
        }
    catch(e){
        }
});


server.post("/api/messages", async (req, res) => {
  console.log("Reached")
  await notificationApp.requestHandler(req, res, async (context) => {
    await teamsBot.run(context);
  });
});

server.listen(process.env.port || process.env.PORT || 3978, () => {
  console.log(`\nApp Started, ${server.name} listening to ${server.url}`);
```

```
});
```

Messaging Endpoint

When a user installs the Bot in Teams App, these steps take place

- **Teams Client Sends an Event Payload:** Teams client creates the bot installation event and passes it to the Bot connector Service through a channel
- **Connector Receives the Message:** The channel sends the message to the Bot Framework Connector.
- **Connector Forwards Message:** The connector processes the message and forwards it to your bot's web service endpoint.
- **Bot Processes Message:** Your bot receives the message, processes it, and determines a response.
- **Response Sent Back:** The bot sends the response back to the connector, which then forwards it to the channel to be delivered to the user. This could be a welcome message for the user

We have the option to disable the ability to send messages to the bot from Teams Application. But messaging endpoint is still needed to capture Conversation References after bot installation. Security of The Messaging Endpoint and Request Handler of the Bot is handled by the Bot-Framework-SDK internally. Bot-Framework-SDK handles authorization by validating the bearer token provided by the Azure Bot Service.

By design, Request Handler of the Conversation Bot created using Bot-framework SDK handles the requests that are coming to this endpoint. This handler performs the authorization and verifies that the requests being processed are legitimate

- **Connector Service Validation**
  - When the Bot Framework Connector sends requests to your bot, it includes a JWT in the Authorization header.
  - The SDK automatically handles the validation of this token. It checks the token's signature, expiry, issuer, and audience to ensure that the request is coming from a legitimate source.
- **Default Middleware**
  - The Bot Framework SDK comes with built-in middleware that processes incoming requests. This middleware is responsible for extracting and validating the JWT from the Authorization header.
  - If the token is valid, the middleware allows the request to proceed to your bot's logic. If not, the request is rejected.
- **Configuration**
  - Typically, when you create a bot using the Microsoft Bot Framework, you configure it with a Microsoft App ID and a password (Microsoft App Password). These credentials are used by the framework to authenticate and validate requests automatically.
  - This means that if your bot is correctly configured with these credentials, the authentication process is handled seamlessly in the background.

Claims in the Token passed from Teams App to Bot Framework Connector

```
{
  "aud": "https://ic3.teams.office.com",
  "iss": "https://sts.windows.net/08f0fb12-6e78-45a9-8166-b225a4a184ea/",
  "iat": 1723409778,
  "nbf": 1723409778,
  "exp": 1723496478,
  "acct": 0,
```

```
  "acr": "1",
  "aio": "AVQAq/8XAAAA......NZciMgHaV+.........+nuKT922AqOob4pz+Y=",
  "amr": [
    "pwd",
    "mfa"
  ],
  "appid": "5e3ce6c0-2b1f-4285-8d4b-75ee78787346",
  "appidacr": "0",
  "given_name": "Sachin",
  "idtyp": "user",
  "ipaddr": "103.....3",
  "name": "Sachin",
  "oid": "da5b38a5-1ede-4253-b26f-abf3ce1ee1ad",
  "puid": "10032003A6588931",
  "rh": "0.AbcAEvvwCHhuqUWBZrIlpKGE6lTwqjmlgcdIpPgCkwEglbn8ALM.",
  "scp": "Teams.AccessAsUser.All",
  "sub": "r3Ayj7edx_qPEfFmkYS76gbeJIfrz2_Hgf-Kwf8vY9c",
  "tid": "08f0fb12-6e78-45a9-8166-b225a4a184ea",
  "unique_name": "sachin@50clbl.onmicrosoft.com",
  "upn": "sachin@50clbl.onmicrosoft.com",
  "uti": "lhSWcOHDqUKVnFgbREFNAA",
  "ver": "1.0",
  "xms_cc": [
    "CP1"
  ],
  "xms_idrel": "12 1",
  "xms_ssm": "1"
}
```


Claims in the Token passed to the bot from the bot framework connector when user sends message to the Bot Expand source


```
{
  "serviceurl": "https://smba.trafficmanager.net/amer/",
  "nbf": 1723411961,
  "exp": 1723415561,
  "iss": "https://api.botframework.com",
  "aud": <bot-id>
}
```


Since we're only interacting with the Bot Framework service and not using any custom APIs or external services that require secure access, then the built-in authentication provided by the Bot Framework SDK should be sufficient.

SSL Certificate and HTTPS Endpoints

Microsoft Bot Framework requires us to provide an endpoint for the messaging endpoint in Bot Framework Portal.

The provided endpoint,

- Should be a HTTPS endpoint
- Should not be an IP address (should have a DNS name)

To have a nodejs restify server serve HTTPS endpoints,

- Obtain a DNS Certificate and the associated private key, either by Obtaining one from a third party or by creating one using
- *openssl req -nodes -new -x509 -keyout server.key -out server.cert*
- Using the obtained certificate and private key, make restify serve https service

```
const fs = require('fs')
const httpsOptions = {key: fs.readFileSync('ssl/key.pem'),
                      certificate: fs.readFileSync('ssl/cert.pem')}
const server = restify.createServer(httpsOptions)
```

If the Virtual Machine running the backend logic does not have a DNS name, we could use reverse proxy services to obtain a DNS name

Storage Considerations

A proactive bot in Microsoft Teams may need to send messages to users even when those users have not directly interacted with the bot recently. To do this, the bot needs to maintain a record of the conversation references. These references allow the bot to locate and re-engage in the previously established conversation or start a new one with the user.

Conversation references contain contextual information about the chat or channel where the bot has interacted with users. This includes:

- **Conversation ID:** Unique identifier for the conversation.
- **User ID:** Identifier for the user.
- **Tenant ID:** Identifier for the tenant.

This contextual information is necessary to send messages or updates to the correct conversation or channel.

Conversation Reference

```
{
  "activityId":"f:4c06e7be-31d2-27d3-2c3f-e2c*****0a",
  "user":{
    "id":"29:xxx",
    "aadObjectId":"xxx"
  },
  "bot":{
    "id":"28:<bot-id>",
    "name":"Siebel Bot"
  },
  "conversation":{
    "conversationType":"personal",
    "tenantId":"<tenant-id>",
    "id":"xxx"
  },
  "channelId":"msteams",
  "locale": "en-GB",
  "serviceUrl":"https://smba.trafficmanager.net/amer/"
}
```

Storage Methods

Azure Storage (Recommended)

Microsoft Bot Framework SDK provides support for keeping the conversation references in Cloud Storage such as Azure Storage.

Cloud Storage is recommended over Local Storage, as local storage does not provide Reliability, Scalability, Security, Backup and Recovery

- Go To Azure Portal. From All Services, choose a new Storage Service
- Create a new storage and create a new Table in the storage
- From Data Storage → Tables, note down the URL(Ignore the '/<table_name>' part at the end) and the table name

- Choose Security + Networking → Access Keys, Note down the Storage Access key and the Storage name

Implementing the methods to perform actions in Azure Table storage

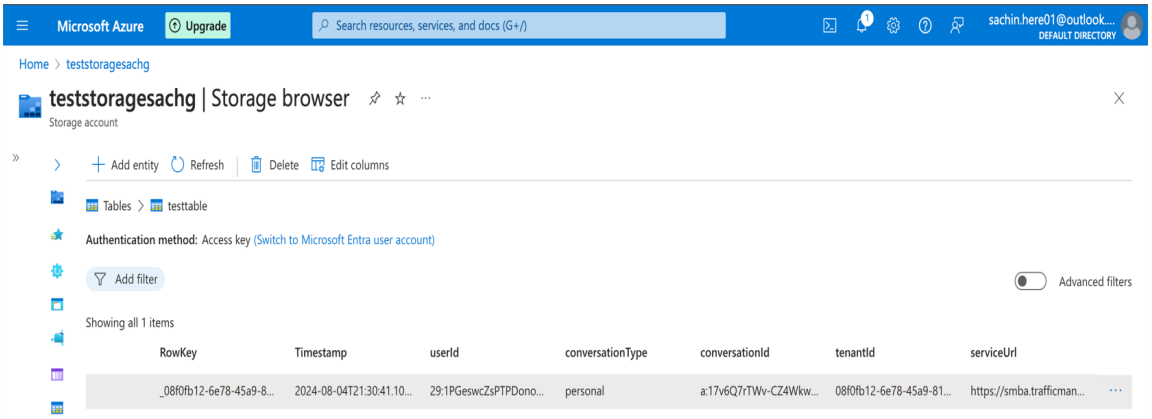
```
const { AzureNamedKeyCredential, TableClient } = require("@azure/data-tables");
const {
    constructConversationReference,
    extractKeyDataFromConversationReference,
} = require("./util");

class TableStore {
    constructor(storageAccountName, storageAccountURL, storageAccountKey, storageTableName){
        const sharedKeyCredential = new AzureNamedKeyCredential(storageAccountName,
storageAccountKey);

        this.client = new TableClient(`${storageAccountURL}`, `${storageTableName}`,
sharedKeyCredential, { allowInsecureConnection: true });
    }
//add conversation reference logic
    async add(key, reference, options) {
        const task = {partitionKey: this.hash(key), rowKey: key,
...extractKeyDataFromConversationReference(reference),};
        try {
            await this.client.createEntity(task);
            return true;
        } catch (e) {
            return false;
        }
    }
// remove conversation reference logic
    async remove(key, reference {
        try {
            await this.client.deleteEntity(this.hash(key), key);
            return true;
        } catch (e) {
            return false;
        }
    }
//list conversation reference logic
    async list(pageSize, continuationToken) {
        const entities = await this.client.listEntities()
                        .byPage({ maxPageSize: pageSize, continuationToken:
continuationToken })
                        .next();

        return {
               data: entities.value.map((entity) => {
                      return constructConversationReference(entity);
                }),
            continuationToken: entities.value.continuationToken,
        };
    }
}

module.exports = TableStore;
```

Initialize the Bot to access Azure Table Storage

```
const { BotBuilderCloudAdapter } = require("@microsoft/teamsfx");
const ConversationBot = BotBuilderCloudAdapter.ConversationBot;
const config = require("./config");
const TableStore = require('../storage') // The above code block

const notificationApp = new ConversationBot({
```

```
  adapterConfig: {
    MicrosoftAppId: config.botId,
    MicrosoftAppPassword: config.botPassword,
    MicrosoftAppType: "MultiTenant",
  },
  notification: {
    enabled: true,
       //Table storage configuration
    store: new TableStore("teststoragesachg",
"https://teststoragesachg.table.core.windows.net",
"2lcbfGr*******StorageAccountKey********bA==", "testtable")
  },
});

module.exports = {
  notificationApp
};
```

Figure 24 - Azure Storage



Local Storage

Local storage is the default method provided by the bot-framework-sdk

Data is stored as an object in,

- *.notification.localstore.json* if running locally.
- *${process.env.TEMP}/.notification.localstore.json*, if *process.env.RUNNING_ON_AZURE* is set to 1.

This method is not recommended for production scenario

Initialize the Bot to access Local Storage

```
const { BotBuilderCloudAdapter } = require("@microsoft/teamsfx");
const ConversationBot = BotBuilderCloudAdapter.ConversationBot;
const config = require("./config");

const notificationApp = new ConversationBot({
  adapterConfig: {
    MicrosoftAppId: config.botId,
    MicrosoftAppPassword: config.botPassword,
    MicrosoftAppType: "MultiTenant",
  },
  // Enable notification
  notification: {
    enabled: true
  },
});
```

```
module.exports = {
  notificationApp,
};
```

Siebel

Flow 1 - Setup

We are using Siebel Server Scripts to invoke the Notification API of the Custom Bot. We will be deploying a tomcat service in Siebel to handle the authentication and to act as a middleware

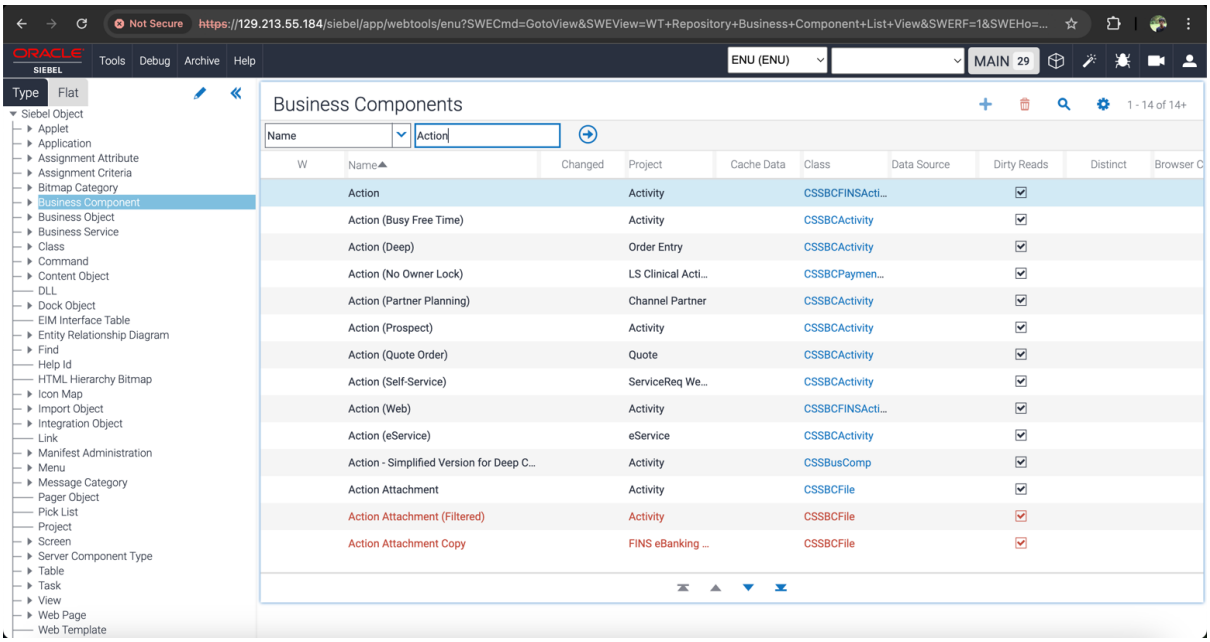


Figure 12 - Siebel Webtools - Action Business Component

Siebel Server Script code

```
function BusComp_WriteRecord() {
try {

  var httpService = TheApplication().GetService("EAI HTTP Transport");
  var httpInputs = TheApplication().NewPropertySet();
  var httpOutputs = TheApplication().NewPropertySet();
  var notificationEndpointUrl = "https://public-ip/dns-of-vm-hosting-the-node-
service:3978/api/notification";

  var description = this.GetFieldValue("Description");
  var priority = this.GetFieldValue("Priority");

  var result = ContinueOperation;
  this.ActivateField("Employee Email Address")

  var email = this.GetFieldValue("Employee Email Address")
  var accountName = this.GetFieldValue("Account Name");
  var activityId = this.GetFieldValue("Id")
  this.ActivateField("Start Time")
  var start = this.GetFieldValue("Start Time")
  var duration = this.GetFieldValue("Duration")
  var due = this.GetFieldValue("Due")
  var activityId = this.GetFieldValue("Id")
```

```
  var deeplink = [Base-Siebel-Url-for-your-application] +
'?SWECmd=GotoView&SWEView=Activity+List+View&SWERF=1&SWEHo=&SWEBU=1&SWEApplet0=Activity
+List+Applet+With+Navigation&SWERowId0='+ activityId;
  var msEmail = email;
  var uid = email.split("@")[0];
  if(uid != ""){
    var msEmail = uid + "@custom email domain for your azure tenancy"; // special
handling for changing the email ids from Siebel to follow the custom email domain for
your azure tenancy

    var requestBody ='{"user":"'+ msEmail+
                     '","accountName":"'+accountName+
                     '","time":"'+ start +
                     '","deepLinkUrl":"'+ deeplink +
                     '","type":"'+ "Activity" +
                     '","description":"'+ description +
                     '","priority":"'+ priority +'"}'

    httpInputs.SetProperty("HTTPRequestURLTemplate", notificationEndpointUrl);
    httpInputs.SetProperty("HTTPRequestMethod", "POST");
    httpInputs.SetProperty("HTTPContentType", "application/json");
    httpInputs.SetProperty("CharSetConversion", "UTF-8");
    httpInputs.SetProperty("HTTPAccept", "*/*");
    httpInputs.SetValue(requestBody);
    httpService.InvokeMethod("SendReceive", httpInputs, httpOutputs);
    }
  } catch (e) {
      TheApplication().SetProfileAttr("Error2", "Error in InvokeREST: " +
e.toString());
      return (CancelOperation);
  }
}
```

Tomcat Middleware Java Code

Tomcat middleware retrieves the authentication token from the identity provider securing the MS Teams Custom Bot Endpoint and forwards it, along with the incoming payload from Siebel to the Bot Endpoint. This middleware serves as an abstraction layer, simplifying authentication and request handling.

```
package com.siebel.external;
import javax.validation.constraints.Null;
import javax.ws.rs.*;
import javax.ws.rs.client.*;
import javax.ws.rs.core.*;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.glassfish.jersey.client.ClientConfig;
import org.glassfish.jersey.client.ClientProperties;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;

@Path("/eventPusher")
public class SiebelExternalService {

    ObjectMapper mapper = new ObjectMapper();
```

```
    @POST
    @Path("/pushEvent")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response pushEvent(String payload, @Context HttpHeaders headers) {

        try {
            JsonNode payloadJson = mapper.readTree(payload);
            String authHeader = headers.getHeaderString("Authorization"); //'Basic
clientId:clientSecret' in base64
            String authScope =
payloadJson.get("authScope")!=null?payloadJson.get("authScope").asText(null) : null;
            String authUrl =
payloadJson.get("authUrl")!=null?payloadJson.get("authUrl").asText(null) : null;

            String accessToken = null;
            if(!Objects.equals(authUrl, null) && !Objects.equals(authUrl, "")&&
!Objects.equals(authScope, null) && !Objects.equals(authScope, "")){
                try {
                    Response idpResponse = obtainToken(authUrl, authHeader, authScope);
                    int statusCode = idpResponse.getStatus();
                    String idpResponseJson = idpResponse.readEntity(String.class);
                    JsonNode idpResponseNode = mapper.readTree(idpResponseJson);
                    if (statusCode != 200) {
                        return Response.status(statusCode).entity(idpResponseJson).build();
                    }
                    accessToken = idpResponseNode.get("access_token").asText();

                } catch (Exception e) {
                    e.printStackTrace();
                    return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
                            .entity("Error obtaining token: " + e.getMessage())
                            .build();
                }
            }


            JsonNode message = payloadJson.get("message");
            String userid =
payloadJson.get("userid")!=null?payloadJson.get("userid").asText(null):null;
            String notificationEndpointUrl =
payloadJson.get("url")!=null?payloadJson.get("url").asText(null):null;
            Map<String, Object> eventPayload = new HashMap<>();
            System.out.println(message);
            if (message != null) {
                parseJsonNode(message, eventPayload);
            }
            eventPayload.put("user", userid);

            MultivaluedMap<String, Object> eventHeaders = new MultivaluedHashMap<>();
            if(accessToken!= null){
                eventHeaders.add("Authorization", "Bearer "+ accessToken);
            }
            eventHeaders.add("Content-Type", "application/json");
            ClientConfig clientConfig = new ClientConfig();
            clientConfig.property(ClientProperties.PROXY_URI, "http://www-
proxy.us.oracle.com:80");
            Client client = ClientBuilder.newClient(clientConfig);
            WebTarget target = client.target(notificationEndpointUrl);
            String jsonEvent = mapper.writeValueAsString(eventPayload);
            Response response = target.request()
                    .headers(eventHeaders)
                    .post(Entity.entity(jsonEvent, "application/json"));

            int statusCode = response.getStatus();
```

```
            String responseBody = response.readEntity(String.class);

            if (statusCode == 200 || statusCode == 201) {
                return Response.status(statusCode).entity("message ok").build();
            } else {
                return Response.status(statusCode).entity(responseBody).build();
            }

        } catch (Exception e) {
            e.printStackTrace();
            return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Error pushing event: " + e.getMessage())
                    .build();
        }
    }
    public void parseJsonNode(JsonNode node, ObjectNode eventPayload) {
        node.fieldNames().forEachRemaining(field -> {
            JsonNode value = node.get(field);

            if (value.isObject()) {
                // If the value is an object, recursively parse it
                parseJsonNode(value, eventPayload);
            } else if (value.isValueNode()) {
                // If it's a simple value, add it to eventPayload
                eventPayload.put(field, value.asText());
            }
        });
    }
    private static Response obtainToken(String authUrl, String authHeader, String authScope)
{

        Form authForm = new Form();
        authForm.param("grant_type","client_credentials");
        authForm.param("scope",authScope);
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.property(ClientProperties.PROXY_URI, "http://www-
proxy.us.oracle.com:80");
        Client client = ClientBuilder.newClient(clientConfig);
        WebTarget target = client.target(authUrl); //works in both Azure and idcs
        Response idpResponse = target.request()
                .header("Authorization", authHeader)
                .post(Entity.entity(authForm, MediaType.APPLICATION_FORM_URLENCODED));

        return  idpResponse;
    }

}
```

Flow 2 - Setup

Business Component Used for the flow

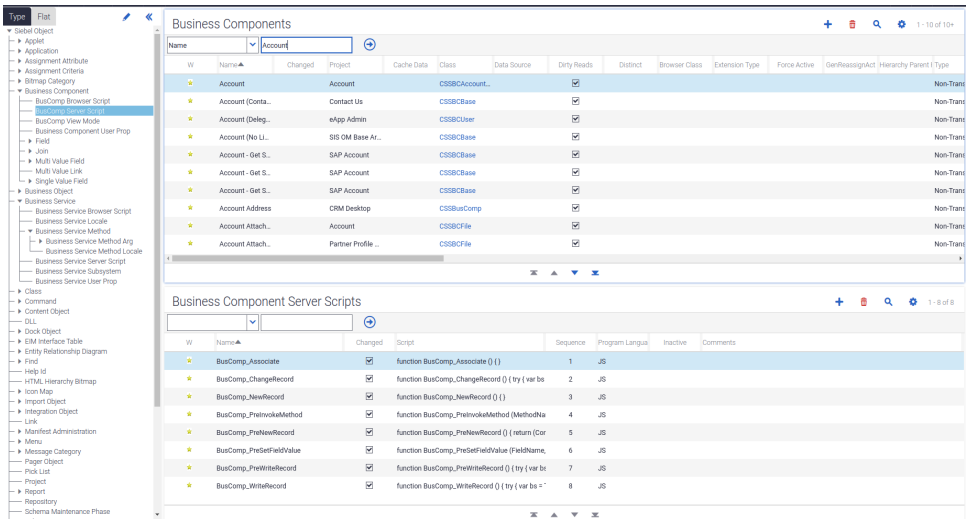


Figure 13 - Siebel Webtools - Account Business Component

Associated Child BC used

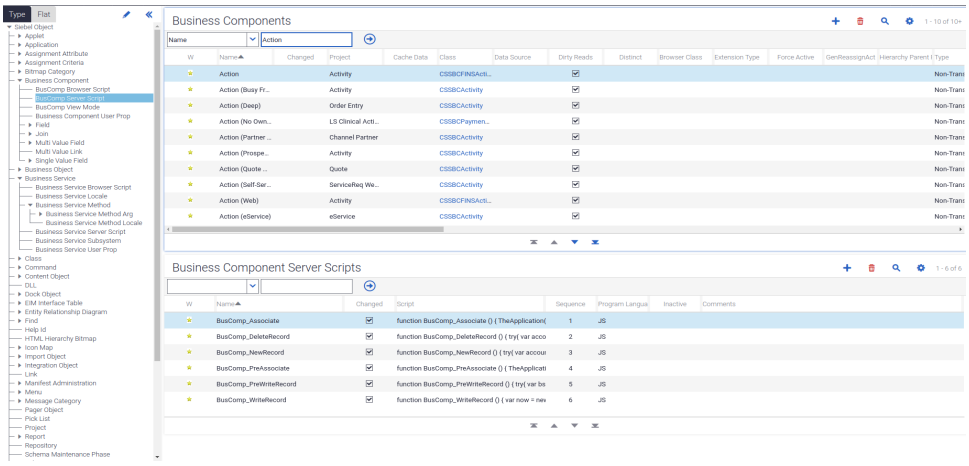


Figure 14 - Siebel Webtools - Action Business Component

Custom Business Service Methods

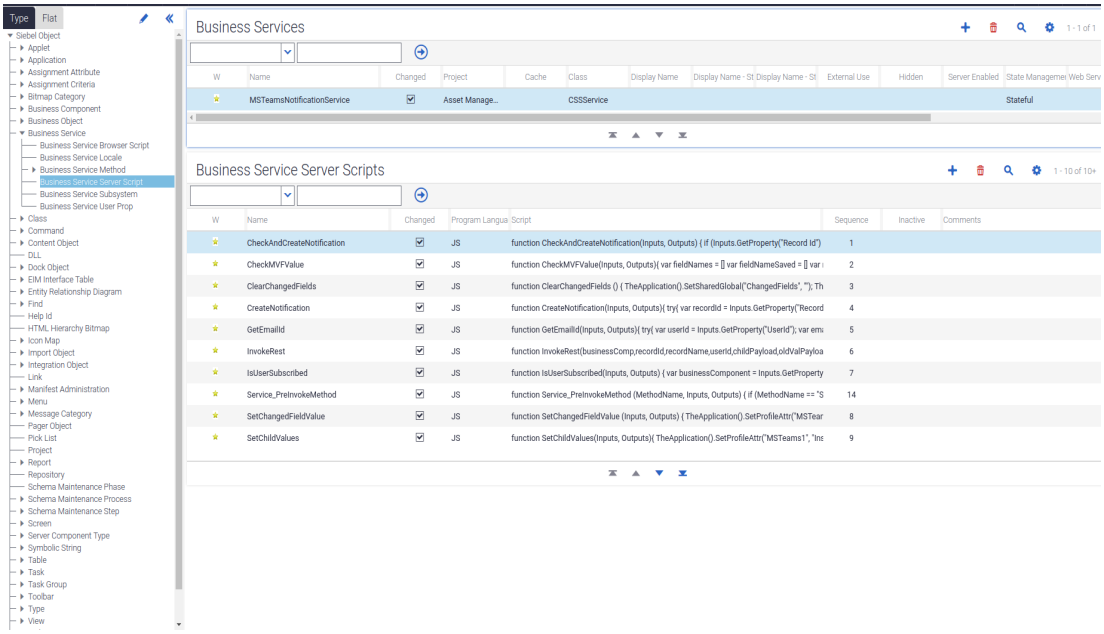


Figure 15 - Siebel Webtools - 'MSTeamsNotificationService' Custom Business Service

Methods are listed and explained in detail below

Custom Business Service PreInvoke Method

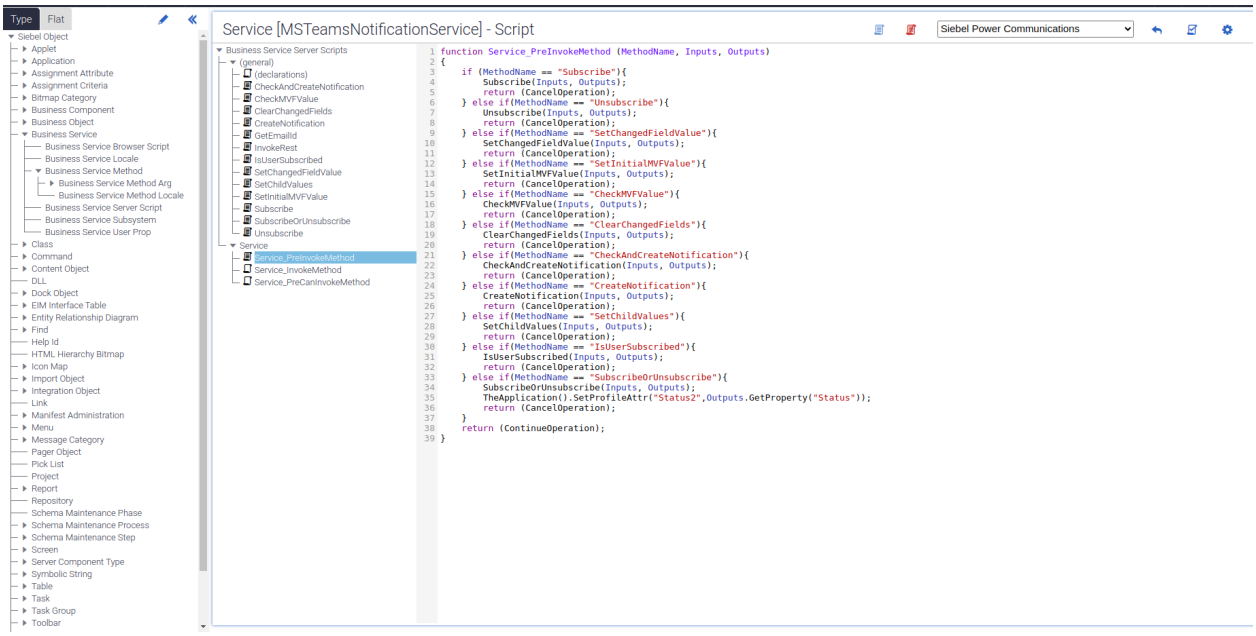


Figure 16 - Siebel Webtools - Custom Business Service PreInvoke Method

Service_PreInvokeMethod

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
    if (MethodName == "Subscribe"){
        Subscribe(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "Unsubscribe"){
        Unsubscribe(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "SetChangedFieldValue"){
        SetChangedFieldValue(Inputs, Outputs);
```

```
        return (CancelOperation);
    } else if(MethodName == "SetInitialMVFValue"){
        SetInitialMVFValue(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "CheckMVFValue"){
        CheckMVFValue(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "ClearChangedFields"){
        ClearChangedFields(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "CheckAndCreateNotification"){
        CheckAndCreateNotification(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "CreateNotification"){
        CreateNotification(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "SetChildValues"){
        SetChildValues(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "IsUserSubscribed"){
        IsUserSubscribed(Inputs, Outputs);
        return (CancelOperation);
    } else if(MethodName == "SubscribeOrUnsubscribe"){
        SubscribeOrUnsubscribe(Inputs, Outputs);
        TheApplication().SetProfileAttr("Status2",Outputs.GetProperty("Status"));
        return (CancelOperation);
    }
    return (ContinueOperation);
}
```

Handling Subscription

We will be handling the subscriptions using *MSTeamsNotificationSubscription* Business Component. We invoke *SubscribeOrUnsubscribe* method of the *MSTeamsNotificationService* Business Service from relevant applets, with the Business Component Name, Record Id, deep link URL and User Id.

Firstly, we add two buttons in the Applet that we are interested in, and we name them as 'Subscribe' and 'Unsubscribe'.

Make sure these buttons are calling the correct Methods - *'subscribeMethod'* and *'unsubscribeMethod'*.

Subscribe/Unsubscribe Invocation from Applet

WebApplet PreCanInvokeMethod Method (Enable/Disable the buttons according to Subscription Status)

```
function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke) {

       if (MethodName == "subscribeMethod" || MethodName == "unsubscribeMethod") {
       try{
            var accountBC = this.BusComp();
            var recordId = accountBC.GetFieldValue("Id");
            var input = TheApplication().NewPropertySet();
            var output = TheApplication().NewPropertySet();
            input.SetProperty("Business Component", accountBC.Name());
            input.SetProperty("User Id", TheApplication().LoginName());
            input.SetProperty("Record Id", recordId);
            var bs = TheApplication().GetService("MSTeamsNotificationService");
            bs.InvokeMethod("IsUserSubscribed",input, output);
              if(output.GetProperty("Status") == "Subscribed"){

       TheApplication().SetProfileAttr(accountBC.GetFieldValue("Id")+"_SubscribedButtonName"
, "Unsubscribe");
                    } else {
```

```
TheApplication().SetProfileAttr(accountBC.GetFieldValue("Id")+"_SubscribedButtonName",
"Subscribe");
            }
            if (MethodName == "subscribeMethod"){
                if(output.GetProperty("Status") == "Subscribed"){
                   CanInvoke = "FALSE";
                } else {
                    CanInvoke = "TRUE";
                }
            } else {
                if(output.GetProperty("Status") == "Subscribed"){
                   CanInvoke = "TRUE";
                } else {
                    CanInvoke = "FALSE";
                }
            }
        } catch (e) {
            return (CancelOperation);
        }
        return(CancelOperation);
    }
    return(ContinueOperation);
}
```

WebApplet PreInvokeMethod (Gets required data and invokes SubscribeOrUnsubscribe)

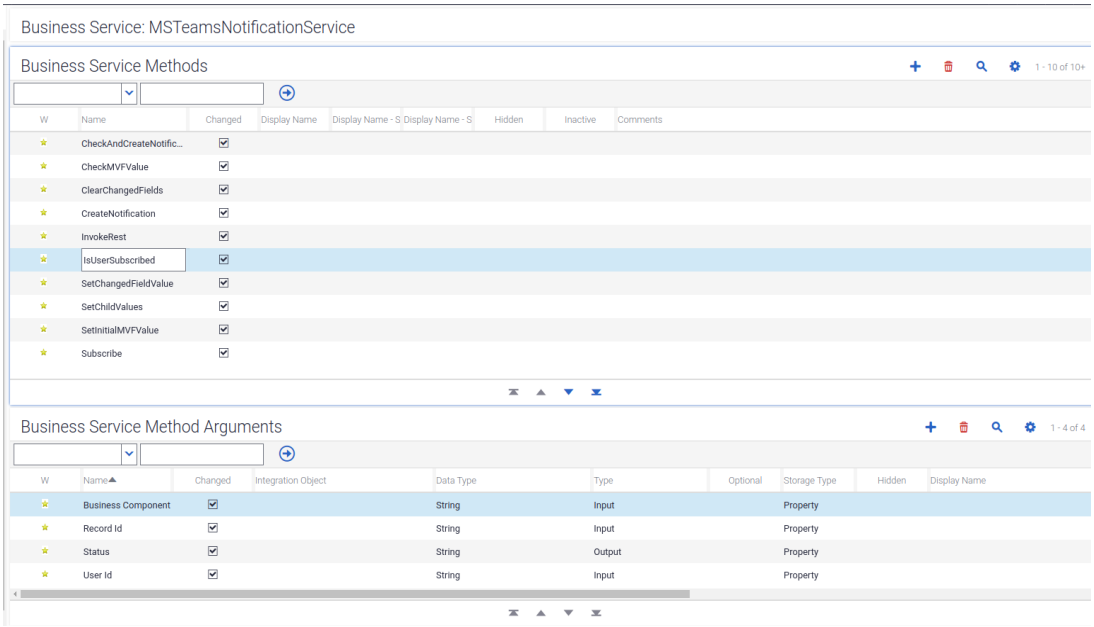
```
function WebApplet_PreInvokeMethod (MethodName)
{
   if(MethodName == "subscribeMethod" || MethodName == "unsubscribeMethod"){

     try{
        var loginName = TheApplication().LoginName();
        var accountBC = this.BusComp();
        var recordId = accountBC.GetFieldValue("Id");

        var bo = TheApplication().GetBusObject("Employee");
        var bc = bo.GetBusComp("Employee");

        var input = TheApplication().NewPropertySet();
        var output = TheApplication().NewPropertySet();

        var viewName = TheApplication().ActiveViewName();
        var appletName = this.Name();
        var busCompName = this.BusComp().Name()
        var baseUrl = "";
        try {
            var appBo = TheApplication().GetBusObject("System Preferences"); // Get the base
url saved as a system preference
            var appBc = appBo.GetBusComp("System Preferences");

            appBc.ActivateField("Value");
            TheApplication().SetProfileAttr("appBc", appBc);
            appBc.ClearToQuery();
            appBc.SetSearchSpec("Name", "Base URL");
            appBc.ExecuteQuery(ForwardOnly);
            if (appBc.FirstRecord()) {
               baseUrl = appBc.GetFieldValue("Value");
               TheApplication().SetProfileAttr("url", baseUrl);
            }
        } catch (e) {
            TheApplication().SetProfileAttr("err", e.toString());
        } finally {
            appBo = null;
            appBc = null;
        }
```

```
        var deeplink = baseUrl? baseUrl + "?SWECmd=GotoView&SWEView=" + viewName +
            "&SWERF=1&SWEHo=&SWEBU=1&SWEApplet0="+appletName+"&SWERowId0="+ recordId: "";

        input.SetProperty("Business Component", busCompName);
        input.SetProperty("URL", deeplink);
        input.SetProperty("User Id", loginName);
        input.SetProperty("Record Id", recordId);

        var bs = TheApplication().GetService("MSTeamsNotificationService");
        bs.InvokeMethod("SubscribeOrUnsubscribe",input, output);

        // Clean up
        bc = null;
        bo = null;
        bs = null;
      } catch (e) {
         TheApplication().SetProfileAttr("error", e.toString());
        return (CancelOperation);
      }
    }
       return (ContinueOperation);
}
```

Business Service Methods

IsUserSubscribed

Figure 17 - MSTeamsNotificationService Custom Business Service - IsUserSubscribed Method

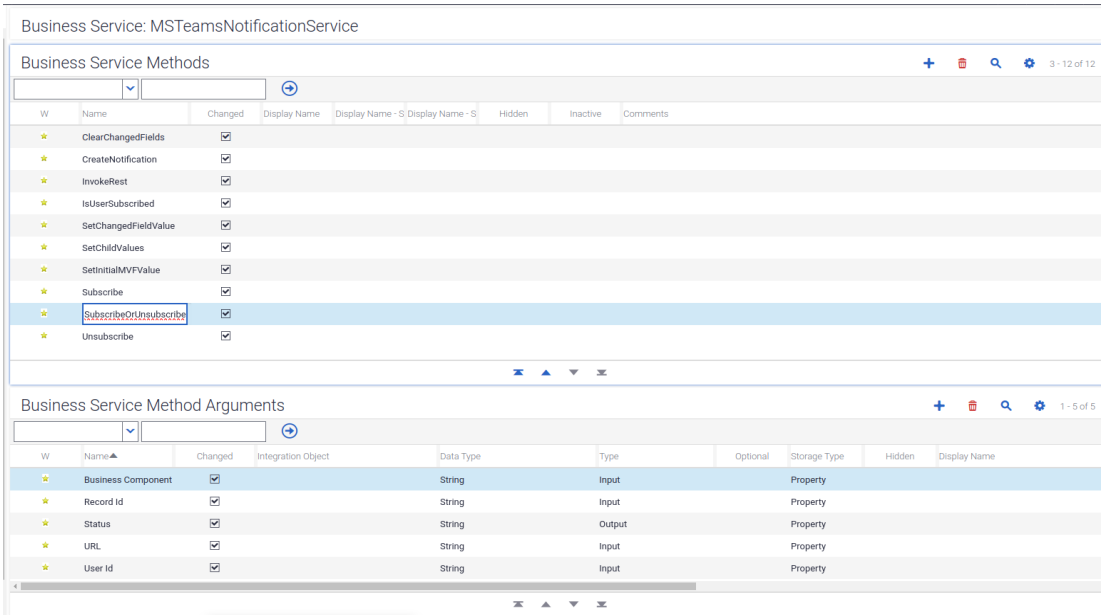



```
function IsUserSubscribed(Inputs, Outputs) {
    var businessComponent = Inputs.GetProperty("Business Component");
    var recordId = Inputs.GetProperty("Record Id");
    var userId = Inputs.GetProperty("User Id");
    var bo = TheApplication().GetBusObject("MSTeamsNotificationSubscription");
    var bc = bo.GetBusComp("MSTeamsNotificationSubscription");
    bc.ClearToQuery();
    bc.SetViewMode(AllView);
    bc.ActivateField("Business Component");
    bc.ActivateField("Record Id");
```

```
    bc.ActivateField("User Id");

    bc.SetSearchSpec("Business Component", businessComponent);
    bc.SetSearchSpec("Record Id", recordId);
    bc.SetSearchSpec("User Id", userId);

    bc.ExecuteQuery(ForwardOnly);
    if (bc.FirstRecord()) {
        Outputs.SetProperty("Status", "Subscribed");
    } else {
        Outputs.SetProperty("Status", "Unsubscribed");
    }
}
```

SubscribeOrUnsubscribe

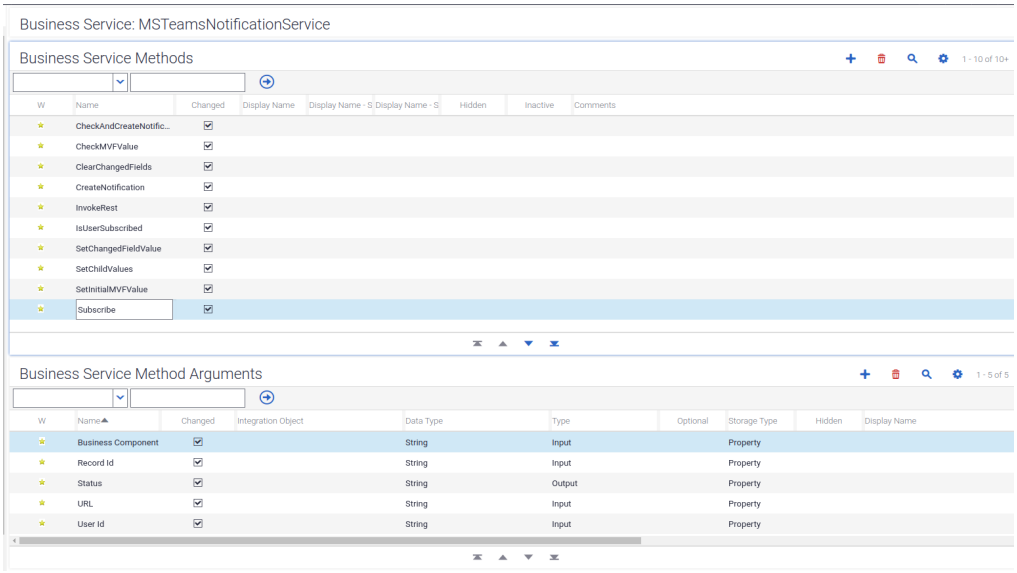
Figure 18 - MSTeamsNotificationService Custom Business Service - SubscribeOrUnsubscribe Method




```
function SubscribeOrUnsubscribe(Inputs, Outputs) {
    IsUserSubscribed(Inputs, Outputs);
    if(Outputs.GetProperty("Status") == "Subscribed"){
        Unsubscribe(Inputs, Outputs);
    } else {
        Subscribe(Inputs, Outputs);
    }
}
```

Subscribe

Figure 19 - MSTeamsNotificationService Custom Business Service - Subscribe Method

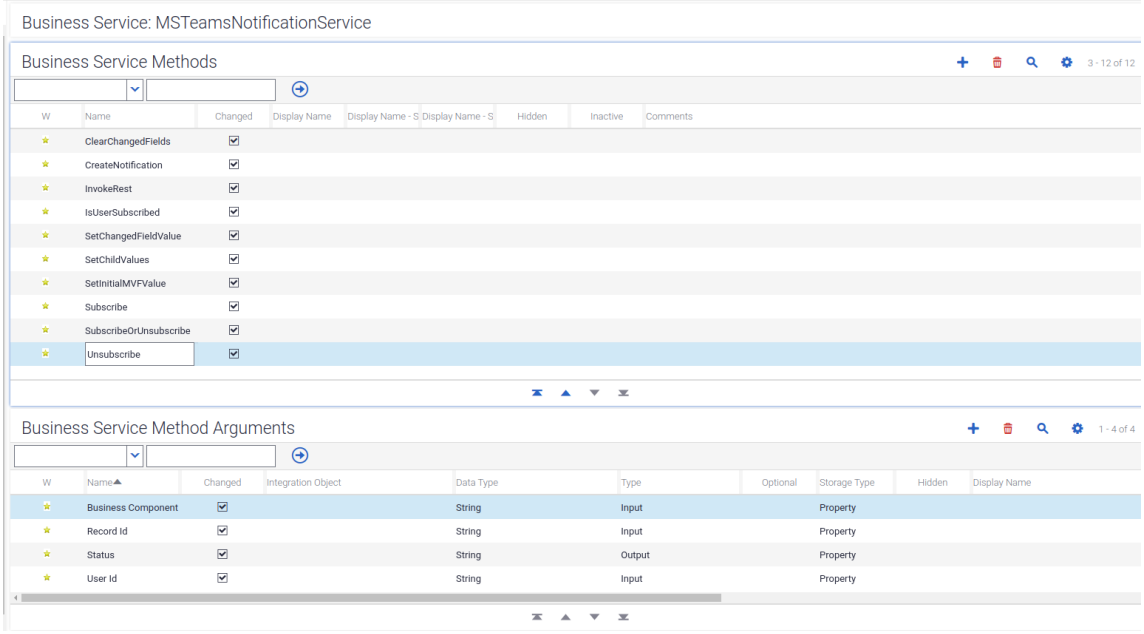



```
function Subscribe(Inputs, Outputs) {
    var businessComponent = Inputs.GetProperty("Business Component");
    var recordId = Inputs.GetProperty("Record Id");
    var userId = Inputs.GetProperty("User Id");
    var url = Inputs.GetProperty("URL");
    var bo = TheApplication().GetBusObject("MSTeamsNotificationSubscription");
    var bc = bo.GetBusComp("MSTeamsNotificationSubscription");
    bc.ClearToQuery();
    bc.SetViewMode(AllView);
    bc.ActivateField("TableRecId");
    bc.ExecuteQuery(ForwardOnly);
    var maxRecId = 0;
    var isRecord = bc.FirstRecord();
    while (isRecord) {
        var currentRecId = parseInt(bc.GetFieldValue("TableRecId"),10);
```

```
        if (currentRecId > maxRecId) {
            maxRecId = currentRecId;
        }
        isRecord = bc.NextRecord();
    }
    var newRecId = maxRecId + 1;

    bc.NewRecord(NewAfter);
    bc.SetFieldValue("Business Component", businessComponent);
    bc.SetFieldValue("URL", url);
    bc.SetFieldValue("Record Id", recordId);
    bc.SetFieldValue("User Id", userId);
    bc.SetFieldValue("TableRecId", newRecId);
    bc.SetFieldValue("Created Date", "01/02/2024");
    bc.WriteRecord();
    Outputs.SetProperty("Status", "Subscribed");
}
```

Unsubscribe

Figure 20 - MSTeamsNotificationService Custom Business Service - Unsubscribe Method




```
function Unsubscribe(Inputs, Outputs) {
    var businessComponent = Inputs.GetProperty("Business Component");
    var recordId = Inputs.GetProperty("Record Id");
    var userId = Inputs.GetProperty("User Id");
    var bo = TheApplication().GetBusObject("MSTeamsNotificationSubscription");
    var bc = bo.GetBusComp("MSTeamsNotificationSubscription");
    bc.ClearToQuery();
    bc.SetViewMode(AllView);
    bc.ActivateField("Business Component");
    bc.ActivateField("Record Id");
    bc.ActivateField("User Id");

    bc.SetSearchSpec("Business Component", businessComponent);
```

```
    bc.SetSearchSpec("Record Id", recordId);
    bc.SetSearchSpec("User Id", userId);

    bc.ExecuteQuery(ForwardOnly);
    if (bc.FirstRecord()) {
        bc.DeleteRecord();
        Outputs.SetProperty("Status", "Unsubscribed");
    } else {
        Outputs.SetProperty("Status", "Not Found");
    }
}
```

Capture the Updates from Siebel & Invoke the CreateNotification Methods

Capturing the field value changes

We capture the field value changes in Siebel using the *BusComp_PreSetFieldValue* event in Siebel. We will call the *SetChangedFieldValue* method of *MSTeamsNotificationService* business service using these update details. *SetChangedFieldValue* sets the values as shared global values, which we can access later. We are writing code on the Account BC

Siebel Business Component Server Scripts - BusComp_PreSetFieldValue

```
function BusComp_PreSetFieldValue (FieldName, FieldValue)
{
    try {
        var bs = TheApplication().GetService("MSTeamsNotificationService");
        var inputs = TheApplication().NewPropertySet();
        inputs.SetProperty("FieldName", FieldName);
        inputs.SetProperty("FieldValue", FieldValue);
        inputs.SetProperty("OldValue", this.GetFieldValue(FieldName));
        var outputs = TheApplication().NewPropertySet();
        bs.InvokeMethod("SetChangedFieldValue", inputs, outputs);
    } catch (e) {
        TheApplication().RaiseErrorText("Error in PreSetFieldValue: " + e.toString());
    }

       return (ContinueOperation);
}
```

Siebel Business Component Server Scripts - BusComp_WriteRecord

```
function BusComp_WriteRecord ()
{
       try {
        var bs = TheApplication().GetService("MSTeamsNotificationService");
        var inputs = TheApplication().NewPropertySet();
        inputs.SetProperty("Record Id", this.GetFieldValue("Id"));
        inputs.SetProperty("Business Component", this.Name());
        inputs.SetProperty("Record Name", this.GetFieldValue("Name"));
        var outputs = TheApplication().NewPropertySet();
        bs.InvokeMethod("CheckAndCreateNotification", inputs, outputs);
    } catch (e) {
        TheApplication().RaiseErrorText("Error in PreSetFieldValue: " + e.toString());
    }
}
```

Custom Business Service Methods

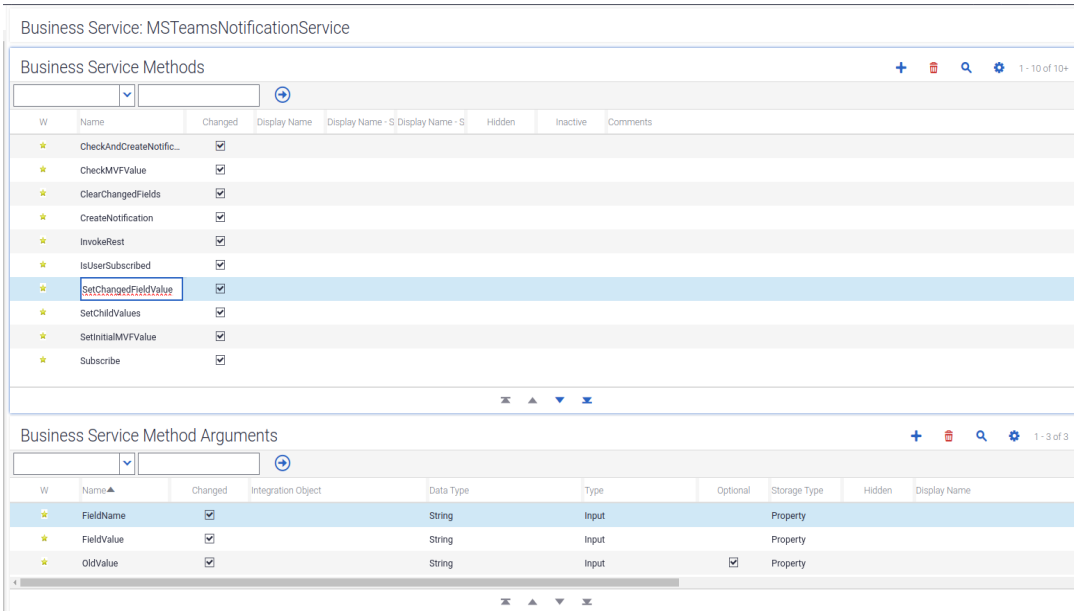


Figure 21 - MSTeamsNotificationService Custom Business Service - SetChangedFieldValue Method

SetChangedFieldValue

```
function SetChangedFieldValue (Inputs, Outputs)
{
    var FieldName = Inputs.GetProperty("FieldName");
    var FieldValue = Inputs.GetProperty("FieldValue");
    var OldValue = Inputs.GetProperty("OldValue");

    var changedFields = TheApplication().GetSharedGlobal("ChangedFields");
    var oldValues = TheApplication().GetSharedGlobal("OldValues");
    var newValues = TheApplication().GetSharedGlobal("NewValues");

    changedFields = changedFields == "" ? '"'+FieldName+'"' :
changedFields+',"'+FieldName+'"';
    oldValues = oldValues == "" ? '"'+OldValue+'"' : oldValues+',"'+OldValue+'"';
    newValues = newValues == "" ? '"'+FieldValue+'"' : newValues+',"'+FieldValue+'"';

    TheApplication().SetSharedGlobal("ChangedFields", changedFields);
    TheApplication().SetSharedGlobal("OldValues", oldValues);
    TheApplication().SetSharedGlobal("NewValues", newValues);

}
```

Capturing the update events of Associated Child Records

We have implemented this functionality in Action BC, which is an Associated Child BC of Account BC. We capture the *BusComp_NewRecord*, *BusComp_PreWriteRecord*, *BusComp_WriteRecord* and *BusComp_DeleteRecord*. To store the update details, we in turn call the *SetChildValues* method of *MSTeamsNotificationService* Business Service. We are writing code on the Action BC and not the parent BC

Siebel Business Component Server Scripts – BusComp NewRecord

```
function BusComp_NewRecord ()
{
    try{

        var accountId = this.GetFieldValue("Account Id");
        if (this.ParentBusComp().Name()=="Account"){
            var bs = TheApplication().GetService("MSTeamsNotificationService");
            var inputsForSetVal = TheApplication().NewPropertySet();
```

```
            inputsForSetVal.SetProperty("ChangedChild", this.Name());
            inputsForSetVal.SetProperty("ChildOperation", "Creation");
            var outputs = TheApplication().NewPropertySet();
            bs.InvokeMethod("SetChildValues", inputsForSetVal, outputs);
            bs = null;
        }
    } catch(e){
        TheApplication().SetProfileAttr("error", accountId + e.toString());
              return (CancelOperation);
    }
}
```

Siebel Business Component Server Scripts – BusComp PreWriteRecord

```
function BusComp_PreWriteRecord ()
{
    try{
        var bs = TheApplication().GetService("MSTeamsNotificationService");

        var accountId = this.GetFieldValue("Account Id");
        var inputsForSetVal = TheApplication().NewPropertySet();
        inputsForSetVal.SetProperty("ChangedChild", this.Name());

if(!TheApplication().GetSharedGlobal("ChildOperation")||TheApplication().GetSharedGlobal("Ch
ildOperation")==''){
              inputsForSetVal.SetProperty("ChildOperation", "Modification");
        }
        var outputs = TheApplication().NewPropertySet();
        bs.InvokeMethod("SetChildValues", inputsForSetVal, outputs);
        bs = null;
    } catch(e){
              return (CancelOperation);
    }
       return (ContinueOperation);
}
```

Siebel Business Component Server Scripts – BusComp WriteRecord

```
function BusComp_WriteRecord ()
{
    try {
        var accountId = this.GetFieldValue("Account Id");  // Get the associated Account Id

        if (accountId != "") {  // Checks if an Account is associated
            var boAccount = TheApplication().GetBusObject("Account")// Get the Account
BusObject
            var bcAccount = boAccount.GetBusComp("Account"); // Get the Account BC

            try{
                var bs = TheApplication().GetService("MSTeamsNotificationService");

                var outputs = TheApplication().NewPropertySet();
                bcAccount.ClearToQuery();
                bcAccount.ActivateField("Id");
                bcAccount.ActivateField("Name");
                bcAccount.SetSearchSpec("Id", accountId);  // Search for the account using
Account Id
                bcAccount.ExecuteQuery(ForwardOnly);
                var inputsForInvoke = TheApplication().NewPropertySet();
                inputsForInvoke.SetProperty("Record Id", accountId);
                inputsForInvoke.SetProperty("Business Component",
this.ParentBusComp().Name());
```

```
                if (bcAccount.FirstRecord()) {

                    inputsForInvoke.SetProperty("Record Name",
bcAccount.GetFieldValue("Name"));
                }
                bs.InvokeMethod("CheckAndCreateNotification", inputsForInvoke, outputs);
                bs = null;
            }
            catch(e){
                TheApplication().SetProfileAttr("error3", e.toString());

            }
        }
    } catch (e) {
        TheApplication().SetProfileAttr("err", e.toString());
        TheApplication().RaiseErrorText("Error in WriteRecord of Action BC: " +
e.toString());
    }
}
```

Siebel Business Component Server Scripts – BusComp DeleteRecord

```
function BusComp_DeleteRecord ()
{
    try{

        var accountId = this.GetFieldValue("Account Id");
        if (this.ParentBusComp().Name()=="Account"){

            var bs = TheApplication().GetService("MSTeamsNotificationService");
            var inputsForSetVal = TheApplication().NewPropertySet();
            inputsForSetVal.SetProperty("ChangedChild", this.Name());
            inputsForSetVal.SetProperty("ChildOperation", "Deletion");
            var outputs = TheApplication().NewPropertySet();
            bs.InvokeMethod("SetChildValues", inputsForSetVal, outputs);
            TheApplication().SetProfileAttr("text", accountId + "deletion");

            var bcAccount = this.ParentBusComp();
            outputs = TheApplication().NewPropertySet();
            bcAccount.ClearToQuery();
            bcAccount.SetSearchSpec("Id",accountId);//Search for the account using Id
            bcAccount.ExecuteQuery(ForwardOnly);
            var inputsForInvoke = TheApplication().NewPropertySet();
            inputsForInvoke.SetProperty("Record Id", accountId);
            inputsForInvoke.SetProperty("Business Component", this.ParentBusComp().Name());
            if (bcAccount.FirstRecord()) {
                inputsForInvoke.SetProperty("Record Name", bcAccount.GetFieldValue("Name"));

            }
            bs.InvokeMethod("CheckAndCreateNotification", inputsForInvoke, outputs);
        }
    } catch(e){

        TheApplication().SetProfileAttr("error3", accountId + e.toString());
        return (CancelOperation);
    }
}
```

Custom Business Service Methods - SetChildValues

```
function SetChildValues(Inputs, Outputs){

    TheApplication().SetProfileAttr("MSTeams1", "Inside SetChildValues");
    var changedChild = Inputs.GetProperty("ChangedChild");
```

```
    var childOperation = Inputs.GetProperty("ChildOperation");

    TheApplication().SetSharedGlobal("ChangedChild", '"'+changedChild+'"');
    if(TheApplication().GetSharedGlobal("ChildOperation")=='' || childOperation ==
"Deletion"){
       TheApplication().SetSharedGlobal("ChildOperation", '"'+childOperation+'"');
    }
}
```

Invocation From Siebel

We are using Siebel Server Scripts to invoke the Notification API of the Custom Bot. We will be storing the 'Follow' details in *'MSTeamsNotificationSubscription'* Business Component. On each update of a record, we capture the update details and then query the Business Component. If there are followers for the record, we obtain their email address and invoke the Notification API of the Custom Bot with a POST payload

POST Payload when Field Changes are captured

```
{
    "user":"tmiller@50clbl.onmicrosoft.com",
    "description": "ABC",
    "modifier": "SADMIN",
    "busComp" : "Account",
    "recName": "3 Com",
    "time": "07/26/2024 03:56:03",
    "oldVal":{"Type":"CME Residential","Account Type Code":"Customer"},
    "newVal":{"Type":"Agency","Account Type Code":"Billing Aggregator"}
}
```

Notification from the Bot:

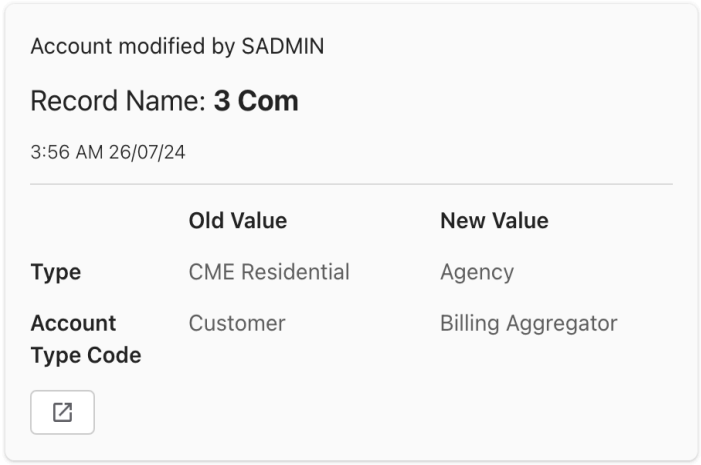


Figure 22 - MS Teams Adaptive card Notification on Account Details modification during Customer Service Agent Flow

POST Payload when Associated Child BC Changes are captured

```
{
    "user":"tmiller@50clbl.onmicrosoft.com",
    "modifier": "SADMIN",
    "busComp" : "Account",
    "recName": "3 Com",
    "time": "07/26/2024 03:56:03",
    "childPayload": { "Action": "Deletion" } // or "Creation" or "Modification"
} // Payload corresponds to when a record from 'Action' BC (Activity BC) associated to the
Account '3 Com' is deleted
```


Confidential - Oracle Restricted

Notification from the Bot:

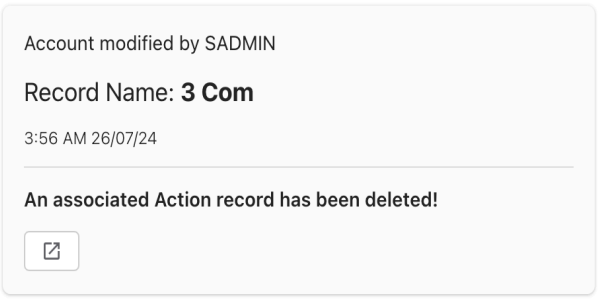


Figure 23 - MS Teams Adaptive card Notification on Account Activity Deletion during Customer Service Agent Flow

Custom Business Service Methods

CheckAndCreateNotification

```
function CheckAndCreateNotification(Inputs, Outputs)
{
    if (Inputs.GetProperty("Record Id") != "") {
        try {
            var notificationService =
TheApplication().GetService("MSTeamsNotificationService");
            var asyncInputs = TheApplication().NewPropertySet();
            var asyncOutputs = TheApplication().NewPropertySet();
            asyncInputs.SetProperty("Record Id", Inputs.GetProperty("Record Id"));
            asyncInputs.SetProperty("Business Component", Inputs.GetProperty("Business
Component"));
            asyncInputs.SetProperty("Record Name", Inputs.GetProperty("Record Name"));
            asyncInputs.SetProperty("User Id", TheApplication().LoginName());
            asyncInputs.SetProperty("ChangedFields",
TheApplication().GetSharedGlobal("ChangedFields"));
            asyncInputs.SetProperty("OldValues",
TheApplication().GetSharedGlobal("OldValues"));
            asyncInputs.SetProperty("NewValues",
TheApplication().GetSharedGlobal("NewValues"));
            asyncInputs.SetProperty("ChangedChild",
TheApplication().GetSharedGlobal("ChangedChild"));
            asyncInputs.SetProperty("ChildOperation",
TheApplication().GetSharedGlobal("ChildOperation"));
            ClearChangedFields();
            notificationService.InvokeMethod("CreateNotification", asyncInputs,
asyncOutputs);
        } catch (e) {
            TheApplication().SetProfileAttr("error", e.toString());
        }
    }
    ClearChangedFields();
}
```

CreateNotification

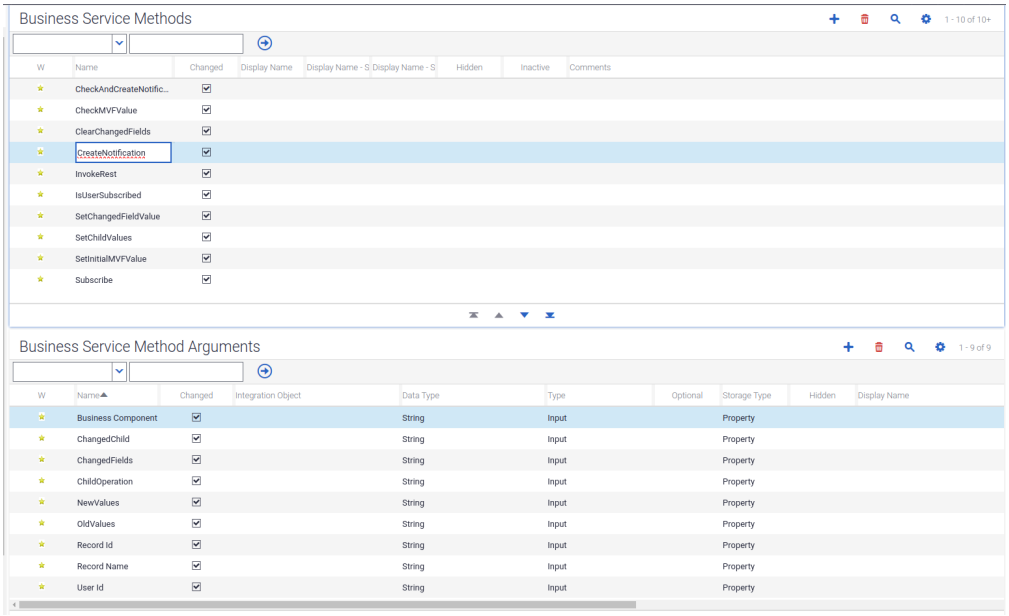


Figure 24 - MSTeamsNotificationService Custom Business Service - CreateNotification Method

```
function CreateNotification(Inputs, Outputs){
    try{
        var recordId = Inputs.GetProperty("Record Id");
        var businessComp = Inputs.GetProperty("Business Component");
        var recordName = Inputs.GetProperty("Record Name");
        var userId = Inputs.GetProperty("User Id");
        TheApplication().SetProfileAttr("recName", recordName);
        var oldValues = []
        var changedFields = []
        var newValues = []
        if(Inputs.GetProperty("OldValues")){
            oldValues = eval('([' + Inputs.GetProperty("OldValues") + '])');
            changedFields = eval('([' + Inputs.GetProperty("ChangedFields") + '])');
            newValues = eval('([' + Inputs.GetProperty("NewValues") + '])');
        }
        var changedChild = []
        var childOper = []
        if(Inputs.GetProperty("ChangedChild")){
            changedChild = eval('([' + Inputs.GetProperty("ChangedChild") + '])');
            childOper = eval('([' + Inputs.GetProperty("ChildOperation") + '])');
        }
        if(changedFields.length > 0){
            var oldValPayload = "{";
            var newValPayload = "{";
            for(var i=0;i<changedFields.length;i++){
                oldValPayload = oldValPayload+'"'+changedFields[i]+'":"'+oldValues[i]+'"';
                newValPayload = newValPayload+'"'+changedFields[i]+'":"'+newValues[i]+'"';
                if(i<changedFields.length-1){
                    oldValPayload = oldValPayload +',';
                    newValPayload = newValPayload +',';
                } else {
                    oldValPayload = oldValPayload +'}';
                    newValPayload = newValPayload +'}';
                }
            }

InvokeRest(businessComp,recordId,recordName,userId,null,oldValPayload,newValPayload)

        }
        if(changedChild.length > 0){
```

```
            var childPayload = "{";
            for(var j=0;j<changedChild.length;j++){
                childPayload = childPayload+'"'+changedChild[j]+'":"'+childOper[j]+'"';
                if(j<changedChild.length-1){
                    childPayload = childPayload +',';
                } else {
                    childPayload = childPayload +'}';
                }
            }
            InvokeRest(businessComp,recordId,recordName,userId,childPayload,null,null);
        }

    } catch(e){
      TheApplication().SetProfileAttr("err1", e.toString());

    }
}
```

GetEmailId

```
function GetEmailId(Inputs, Outputs){
    try{
        var userId = Inputs.GetProperty("UserId");
        var emailId = "";
        var empBo = TheApplication().GetBusObject("Employee");
        var empBc = empBo.GetBusComp("Employee");
        empBc.ActivateField("EMail Addr");
        empBc.ClearToQuery();
        empBc.SetSearchSpec("Login Name", userId);
        empBc.ExecuteQuery(ForwardOnly);
        if (empBc.FirstRecord()) {
            emailId = empBc.GetFieldValue("EMail Addr");

        }
        empBc = null;
        empBo = null;
        Outputs.SetProperty("EmailId", emailId);
    } catch (e) {
        TheApplication().SetProfileAttr("error", e.toString());
    }
    return Outputs;
}
```

Code to Invoke the Custom Bot Endpoint

```
function InvokeRest(businessComp, recordId, recordName, userId, childPayload, oldValPayload,
newValPayload){
        try {

            var bo = TheApplication().GetBusObject("MSTeamsNotificationSubscription");
            var bc = bo.GetBusComp("MSTeamsNotificationSubscription");
            bc.ClearToQuery();
            bc.SetViewMode(AllView);
            bc.ActivateField("Business Component");
            bc.ActivateField("Record Id");
            bc.ActivateField("User Id");
            bc.ActivateField("URL");

            bc.SetSearchSpec("Business Component", businessComp);
            bc.SetSearchSpec("Record Id", recordId);

            bc.ExecuteQuery(ForwardOnly); // fetch the subscription details
```

```
            var isRecord = bc.FirstRecord();
            while (isRecord) {
                var emailInputs = TheApplication().NewPropertySet();
                var emailOutputs = TheApplication().NewPropertySet();
                var followedUid = bc.GetFieldValue("User Id");
                emailInputs.SetProperty("UserId", followedUid);
                GetEmailId(emailInputs, emailOutputs); // function to fetch the email id
from 'Employee' BC by using the user ids
                var emailId = emailOutputs.GetProperty("EmailId");
                var deepLink = bc.GetFieldValue("URL");

                try {
                    var httpService = TheApplication().GetService("EAI HTTP Transport");
                    var httpInputs = TheApplication().NewPropertySet();
                    var httpOutputs = TheApplication().NewPropertySet();
                    var uid = "";
                    if(emailId!= ""){
                        uid = emailId.split("@")[0]+ "@50clbl.onmicrosoft.com";
                        TheApplication().SetProfileAttr("email", emailId);

                    }
                    if(uid!=""){
                        var url = "http://132.**2.**5.***:3978/api/notification";
                        var requestBody = ""
                        if(childPayload!= null){
                           requestBody ='{"user":"' + uid +
                                        '","modifier": "'+userId+
                                        '","busComp" : "'+businessComp+
                                        '","recId" : "'+recordId+
                                        '","recName": "'+recordName+
                                        '","time": "'+new Date()+
                                        '","childPayload":'+childPayload+
                                        ',"deepLinkUrl": "'+deepLink+'"}';
                        } else{
                           requestBody ='{"user":"' + uid +
                                        '","modifier": "'+userId+
                                        '","busComp" : "'+businessComp+
                                        '","recId" : "'+recordId+
                                        '","recName": "'+recordName+
                                        '","time": "'+new Date()+
                                        '","oldVal":'+oldValPayload+
                                        ',"newVal":'+newValPayload+
                                        ',"deepLinkUrl": "'+deepLink+'"}';

                        }

                        httpInputs.SetProperty("HTTPRequestURLTemplate", url);
                        httpInputs.SetProperty("HTTPRequestMethod", "POST");
                        httpInputs.SetProperty("HTTPContentType", "application/json");
                        httpInputs.SetProperty("CharSetConversion", "UTF-8");
                        httpInputs.SetProperty("HTTPAccept", "*/*");
                        httpInputs.SetValue(requestBody);
                        httpService.InvokeMethod("SendReceive", httpInputs, httpOutputs);
                    }

                } catch (e) {
                    TheApplication().SetProfileAttr("error", e.toString());
                    TheApplication().RaiseErrorText("Error in WriteRecord: " +
e.toString());
                }


                isRecord = bc.NextRecord();
            }
        } catch (e) {
```

```
            TheApplication().RaiseErrorText("Error in WriteRecord: " + e.toString());
        }
    }
```

Special Handling

In Siebel, the changes to Multi-Value Field value does not trigger *BusComp_PreSetFieldValue* or *BusComp_SetFieldValue* events. Hence, we need to have special handling in this scenario. We would need to,

1. Store the values of Multi-Valued Fields using the *BusComp_ChangeRecord* event
2. Check for value changes using the *BusComp_PreWriteRecord* event and save these along with other field changes by invoking *SetChangedFieldValue* method of *MSTeamsNotificationService* Business service
3. Replace the initially stored Multi-Valued Fields value with the latest value

Since we are saving the MVF modifications along with other field changes, we don't need to have special handling for the Invocation of Custom Bot API.

We are modifying the Server Scripts on Account BC here

Storing the MVF Fields

```
function BusComp_ChangeRecord ()
{
    try {
        var bs = TheApplication().GetService("MSTeamsNotificationService");
        var inputs = TheApplication().NewPropertySet();
        var outputs = TheApplication().NewPropertySet();

        bs.InvokeMethod("ClearChangedFields", inputs, outputs);

        inputs.SetProperty("FieldName", "Street Address");
        inputs.SetProperty("FieldValue", this.GetFieldValue("Street Address"));
        bs.InvokeMethod("SetInitialMVFValue", inputs, outputs);
    } catch (e) {
        TheApplication().RaiseErrorText("Error in PreSetFieldValue: " + e.toString());
    }
}
```

SetInitialMVFValue

```
function SetInitialMVFValue(Inputs, Outputs){

    var FieldName = Inputs.GetProperty("FieldName");
    var FieldValue = Inputs.GetProperty("FieldValue");

    var mvfFields = TheApplication().GetSharedGlobal("MVFFields");
    var mvfOldValues = TheApplication().GetSharedGlobal("MVFOldValue");

    mvfFields = mvfFields == "" ? '"'+FieldName+'"' : mvfFields+',"'+FieldName+'"';
    mvfOldValues = mvfOldValues == "" ? '"'+FieldValue+'"' :
mvfOldValues+',"'+FieldValue+'"';

    TheApplication().SetSharedGlobal("MVFFields", mvfFields);
    TheApplication().SetSharedGlobal("MVFOldValue", mvfOldValues);

}
```

Check for Value Modifications

BusComp PreWriteRecord

```
function BusComp_PreWriteRecord ()
{
    try {
        var bs = TheApplication().GetService("MSTeamsNotificationService");
```

```
        var inputs = TheApplication().NewPropertySet();
        inputs.SetProperty("FieldNames", '"Street Address"');
        inputs.SetProperty("NewValues", '"'+this.GetFieldValue("Street Address")+'"');
        var outputs = TheApplication().NewPropertySet();
        bs.InvokeMethod("CheckMVFValue", inputs, outputs);
    } catch (e) {
        TheApplication().RaiseErrorText("Error in PreSetFieldValue: " + e.toString());
    }
       return (ContinueOperation);
}
```

CheckMVFValue

```
function CheckMVFValue(Inputs, Outputs){

    var fieldNames = []
    var fieldNameSaved = []
    var newVals = []
    var oldVals = []
  try{
// obtains old values from stored values and new values from input properties
    if(Inputs.GetProperty("FieldNames")){
        oldVals = eval('[' + TheApplication().GetSharedGlobal("MVFOldValue") + ']');
        fieldNameSaved = eval('[' + TheApplication().GetSharedGlobal("MVFFields") + ']');
        fieldNames = eval('[' + Inputs.GetProperty("FieldNames") + ']');
        newVals = eval('[' + Inputs.GetProperty("NewValues") + ']');
    }
    var oldValPayload = {};
    var newValPayload = {};

    if(fieldNames.length > 0){
        for(var i=0;i<fieldNames.length;i++){
            newValPayload[fieldNames[i]] = newVals[i] ? newVals[i] : "";
        }
    }
    if(fieldNameSaved.length > 0){
        for(var j=0;j<fieldNameSaved.length;j++){
            oldValPayload[fieldNameSaved[j]] = oldVals[j] ? oldVals[j] : "";
        }
    }
// obtain all the keys
    var keys = {}

    for(var key1 in oldValPayload){
        if(!keys[key1]){
            keys[key1] = true;
        }
    }

    for(var key2 in newValPayload){
        if(!keys[key2]){
            keys[key2] = true;
        }
    }
// remove unchanged values and handling special cases
    for(var key in keys){
        if(newValPayload[key] == oldValPayload[key]){
            delete newValPayload[key]
            delete oldValPayload[key]
            delete keys[key]
        }
        else if(!newValPayload[key]){
            if(oldValPayload[key]==""){
                delete oldValPayload[key]
                delete keys[key]
            } else {
```

```
            newValPayload[key] ="";
          }
        }
        else if(!oldValPayload[key]){
          if(newValPayload[key]==""){
            delete newValPayload[key]
            delete keys[key]
          } else {
            oldValPayload[key] ="";
          }
        }
    }
// Call SetChangedFieldValue Custom method to store the Multi-Valued Field modification
details along with other Field modifications
    for(var finalkey in keys){

        var inputs = TheApplication().NewPropertySet();
        inputs.SetProperty("FieldName", finalkey);
        inputs.SetProperty("FieldValue", newValPayload[finalkey]);
        inputs.SetProperty("OldValue", oldValPayload[finalkey]);
        var outputs = TheApplication().NewPropertySet();
        SetChangedFieldValue(inputs, outputs);
    }

  } catch(e){
        TheApplication().SetProfileAttr("error3", e.toString())
  }

}
```

Reset the Multi-Valued Field details with the latest ClearChangedFields

```
function ClearChangedFields ()
{
    TheApplication().SetSharedGlobal("ChangedFields", "");
    TheApplication().SetSharedGlobal("OldValues", "");
    TheApplication().SetSharedGlobal("NewValues", "");
    TheApplication().SetSharedGlobal("ChangedChild", "");
    TheApplication().SetSharedGlobal("ChildOperation", "");

// These lines clears the MVF update details
    TheApplication().SetSharedGlobal("MVFFields","");
    TheApplication().SetSharedGlobal("MVFOldValue","");
}

function BusComp_WriteRecord ()
{
       try {
        var bs = TheApplication().GetService("MSTeamsNotificationService");
        var inputs = TheApplication().NewPropertySet();
        TheApplication().SetProfileAttr("val1",this.GetFieldValue("Street Address"));
        inputs.SetProperty("Record Id", this.GetFieldValue("Id"));
        inputs.SetProperty("Business Component", this.Name());
        inputs.SetProperty("Record Name", this.GetFieldValue("Name"));
        var outputs = TheApplication().NewPropertySet();
        bs.InvokeMethod("CheckAndCreateNotification", inputs, outputs);     // internally
calls ClearChangedFields method, which clears the previously stored MVF value from session
storage

// Store the latest MVF value
        var inputSet = TheApplication().NewPropertySet();
        inputSet.SetProperty("FieldName", "Street Address");
        inputSet.SetProperty("FieldValue", this.GetFieldValue("Street Address"));
        var outputSet = TheApplication().NewPropertySet();
```


Confidential - Oracle Restricted

```
        bs.InvokeMethod("SetInitialMVFValue", inputSet, outputSet);

    } catch (e) {
        TheApplication().RaiseErrorText("Error in PreSetFieldValue: " + e.toString());
    }
}
```

Benefits

- **Instant Notifications& Real-time Interaction within MS Teams:** Field agents get immediate notifications about assigned activities. This reduces delays in communication and ensures timely updates. Agents receive instant notifications and updates directly within the MS Teams environment, where they are already working. This allows them to stay informed and collaborate with team members without needing to switch between different platforms.
- **Seamless Access to Siebel Data:** Providing a direct link to the Siebel record from MS Teams allows agents to quickly access relevant information about the task, customer, or case without having to navigate between applications.
- **Enhanced Productivity**: The integration reduces the need for agents to switch between Siebel and MS Teams by providing task notifications and direct links to relevant records, enabling them to prioritize tasks more efficiently and stay on top of important updates without losing focus.

Future Possibilities and Innovations

- **Fetch Data Based on Agent's Needs**: Enable agents to log in to Siebel directly through MS Teams using federated login, allowing them to retrieve and interact with Siebel data seamlessly through simple commands within the Teams environment.
- **Embedded Analytics**: Imagine integrating Siebel's reporting capabilities directly within MS Teams. A field agent could view personalized dashboards or metrics on their performance or upcoming tasks directly within the Teams interface, without needing to access Siebel separately.
- **Integrated Knowledge Base:** The integration could evolve to provide knowledge articles directly in MS Teams based on the case or task at hand. This could help agents quickly solve issues without needing to leave Teams to search the Siebel knowledge base.
- **Voice-Activated Assistance**: With MS Teams already supporting voice commands, agents could interact with the bot using voice commands. They could ask for updates, query Siebel records, or even create follow-up tasks through speech, making the process more hands-free and efficient.

Limitations

- **Potential Downtime:** Since the integration depends on both Siebel and MS Teams, any downtime or connectivity issues on either side could disrupt the flow of notifications or access to Siebel records.
- **Custom Development Challenges:** Since the integration is custom-built, it may require significant effort to maintain, debug, and ensure compatibility with new versions of Siebel or Teams, especially if custom APIs or connectors are used.
- **MS Teams as a Communication Tool:** While MS Teams excels in collaboration and communication, it cannot fully replicate the complex workflows, reporting, and analytics capabilities of Siebel. While Teams can be used for task updates and interactions, more advanced Siebel processes and flows will still need to be handled directly within Siebel.
- **Compliance Risks:** If the integration isn't properly managed, there could be compliance risks, especially in industries with strict data privacy regulations (e.g., healthcare, finance). Ensuring that sensitive Siebel records are handled in accordance with legal and regulatory requirements is essential.

Confidential - Oracle Restricted

Open Items/ Known Issues

- Connecting Oracle Autonomous DB as Storage - OPEN
- The notification at the user end only shows 'Siebel Bot sent a card' and not the details of the notification - forum thread
- Edge cases& Error handling in Siebel - OPEN

References

Design

- Bot Builder Basics
- Bot Activity Handler Concept
- Proactive Installation using Graph

Security

- Configure Authentication in sample node web app with api
- https://github.com/Azure-Samples/ms-identity-javascript-react-tutorial/tree/main/3-Authorization-II/1-call-api
- what-keeps-my-bot-secure-from-clients-impersonating-the-bot-framework-service
- bot-framework-rest-connector-authentication
- https://bitmapbytes.com/idcs-application-oauth2-authentication-with-oracle-content-management/
- https://learn.microsoft.com/en-us/entra/architecture/auth-oauth2

Storage

- https://learn.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/interactive-notification-bot-in-teams
- https://learn.microsoft.com/en-us/javascript/api/@microsoft/teamsfx/botbuildercloudadapter.notificationoptions?view=msteams-client-js-latest
- https://github.com/OfficeDev/teams-toolkit-samples/tree/dev/large-scale-notification

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle