Siebel

Siebel AI Guide

August 2026

Siebel
Siebel AI Guide

August 2026

G59520-01

Author: Tejaswi Tatavarthi

Contents

Get Help

In addition to the resources available in Oracle Help Center, you can explore training, connect with the Oracle community, and share your feedback.

Get Training

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

Join Our Community

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

Share Your Feedback

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

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

Thanks for helping us improve our user assistance!

1 What's New

What's New in This Release

This is the initial release of this guide.

2 Classic AI

Overview of Artificial Intelligence (AI) Services for Siebel CRM

This chapter describes how to set up and configure Oracle Cloud Infrastructure (OCI) AI Services for Siebel CRM. It also includes some sample Artificial Intelligence (AI) use cases.

It includes these topics:

- *About Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services*
- *Overview of Artificial Intelligence (AI) Services for Siebel CRM*
- *Accessing Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services*
- *Process of Setting Up Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services*
- *Artificial Intelligence (AI) Services Use Cases*
- *Siebel REST APIs for Artificial Intelligence (AI) Services*

About Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services

Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services is a collection of services with prebuilt machine learning models that make it easier for developers to apply AI to applications and business operations. Using OCI AI Services make it possible for developers to easily add machine learning to applications without slowing down application development.

The fully managed AI services include:

- **OCI GenAI.** Helps users build Generative AI use cases with underlying Large Language Models(LLMs). Refer the GenAI section for complete details
- **OCI Vision.** Provides pre-trained computer vision models for image recognition and document analysis tasks.

Integration of the following OCI AI Services with Siebel CRM is currently supported:

- **OCI Language**, which is an AI service that applies AI and sophisticated text analysis to understanding textual information.

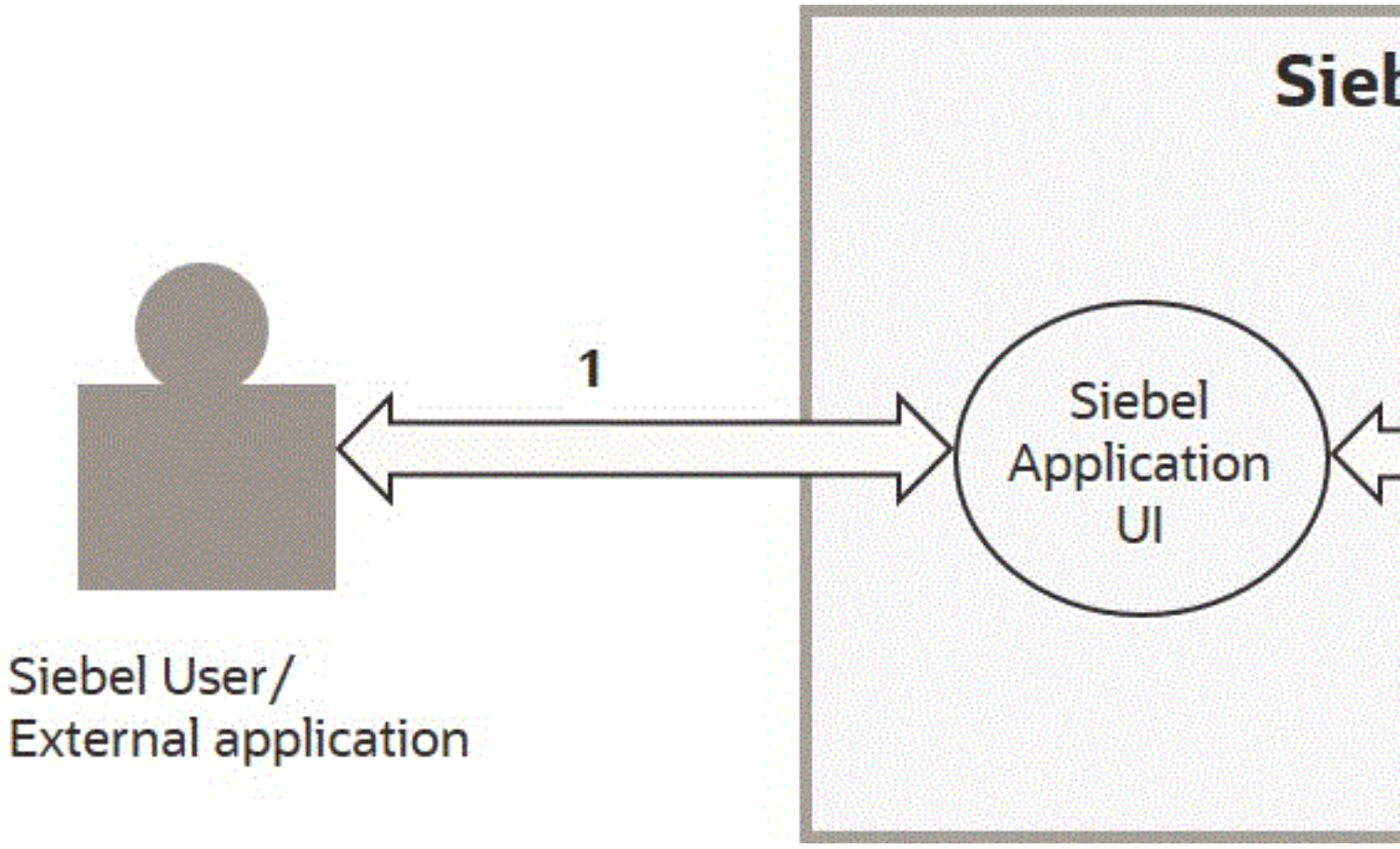
  The OCI Language AI service performs text analysis at scale to understand the unstructured text in documents, customer interactions, and support tickets. It does this by leveraging named entity recognition (NER) to identify specific entities such as names of people, organizations, locations, dates, and times. This service supports the *Detect Personal Identifiable Information (PII) and Private Health Information (PHI)* use case. The masking options supported are –Mask, Replace, or Remove. For more information, see *Use Case 1: Identify and Flag PII/PHI in Service Request Description*.
- **OCI Speech**, which uses automatic speech recognition to convert speech/audio files to text transcriptions.

The OCI Speech AI service converts file-based audio data containing human speech into highly accurate text transcriptions, and can be used to enhance the analysis of audio content. Some of the supported languages are English, Spanish, French, German, Italian. For the complete list of supported languages for different models, refer the OCI documentation. In addition to Oracle Speech model, the OCI Speech service now supports the Whisper model from OpenAI. For more information, see *Use Case 2: Transcribe Customer Calls and Attach Text File to Service Request*.

Accessing Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services

There are three ways to use and access OCI AI Services:

1. You can use OCI AI Services via Siebel Application as shown in the following *reference architecture* for OCI AI Services.
2. You can use OCI AI Services directly via Siebel REST API – for more information, see *Import Supplier Sites*.
3. You can use OCI AI Services via customer components in Siebel CRM.



Reference architecture for OCI AI Services

As shown in this reference architecture, a typical flow for using OCI AI Services is as follows:

1. On the Siebel CRM side:
    - The Siebel CRM client (user or external application) invokes the Siebel OCI AI Service via the Siebel UI.
    - Siebel OCI AI REST API can also be directly invoked via any Siebel component (such as a Workflow) or via any external self-service application layer.
2. On the OCI side:
    - Siebel REST API completes the required pre-processing, authenticates with OCI, and invokes the OCI AI Service via OCI Java SDK to get the required response using AI/ML (Machine Language).
    - When the post-processing completes, the required response is returned to the client.
    - OCI AI Services also use the OCI Object Storage for certain use cases (for example, especially for Speech).

Process of Setting Up Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services

To set up and configure OCI AI Services, perform the following tasks.

- *Load Data into Tables*
- *Configuring OCI Generative AI Service Integration using Template File*
- *Get Started with Sales Territories and Assignment*
- *Uptake of AI/ML powered features with OCI Java SDK 3.54.0*
- *Configure Seed Data and Business Service Access*
- *Configure the Oracle Cloud Infrastructure (OCI) Language AI Service*
- *Configure Seed Data and Business Service Access*

Enable Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services

Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) services are not enabled by default. To use OCI AI services, you must enable certain system preferences as shown in the following procedure.

Configure the system preferences as shown in this table.

System Preference	Value
EnableOCIAILanguage	Set to Y to enable OCI Language AI Service. Set to N to disable OCI Language Service.
EnableOCIAISpeech	Set to Y to enable OCI Speech AI Service. Set to N to disable OCI Speech Service.

System Preference	Value

Configuring OCI Generative AI Service Integration using Template File

This topic describes how to use the `ociconfig-template.properties` template file to configure OCI Generative AI integration, enable new Gen AI features, and manage configuration updates during installation and upgrades.

The `ociconfig-template.properties` template file contains the latest predefined configuration required to support new Gen AI functionalities.

New Customers – Fresh Installation

- The `ociconfig-template.properties` template file is delivered as part of the installation.
- No `ociconfig.properties` file is created by default.
- You must do the following:
  a. Copy and rename `ociconfig-template.properties` to `ociconfig.properties`.
  b. Modify `ociconfig.properties` as required.
  c. Use this file to enable and configure new Gen AI features.

New Customers – Subsequent Upgrades After Initial Installation

New Customers – Fresh Installation (subsequent upgrades with/without new feature changes)

- The `ociconfig-template.properties` template file is delivered as part of the installation.
- No modification/update to the `ociconfig-template.properties` template file is required.
- If any future modifications are required in `ociconfig.properties` due to changes in `ociconfig-template.properties`, you must carefully review the updates and manually merge the necessary changes into `ociconfig.properties` by referring to the `ociconfig-template.properties` file.

Existing Customers – Upgrading to 26.6 and Later

Scenario 1: Upgrade from pre-26.6 to utilize new features

- The `ociconfig-template.properties` template file is newly introduced and copied during the upgrade.
- The existing `ociconfig.properties` template file will be deleted during the upgrade.
- You must do the following:
  a. Take a backup of their existing `ociconfig.properties` file before the upgrade.
  b. After the upgrade, restore the file as `ociconfig.properties`.
  c. Modify/update `ociconfig.properties` to align with the new template.

> **Note:** This is a one-time impact during the upgrade to 26.6 (or later).

Scenario 2: Subsequent Upgrades (26.6 - 26.10 and later)

- The `ociconfig-template.properties` template file is replaced with the latest OX9 version during each upgrade.

- The existing `ociconfig.properties` file:
  - Is not modified or deleted.
  - Is treated as a customer-managed configuration file.
- You must do the following:
  a. Review changes in the updated `ociconfig-template.properties`.
  b. Manually merge any required changes into `ociconfig.properties` by referring to `ociconfig-template.properties`.

Configure Authentication for Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) Services

Complete the steps in the following procedure to configure authentication for Oracle Cloud Infrastructure (OCI) Artificial Intelligence (AI) services.

1. Download the configuration file and the .pem file from OCI.

   In the configuration file, make a note of the credential details for the following parameters: user/userId, region, fingerprint, compartmentId, and tenancy.
2. Navigate to Siebel's internal Tomcat's webapps folder:
   - On Windows: `"<siebel home path>\ses\applicationcontainer_internal\webapps"`

     ```
     // For example:
     "C:\2022_06\ses\applicationcontainer_internal\webapps"
     ```
   - On Unix: `"<siebel home path>/ses/applicationcontainer_internal/webapps"`

     ```
     // For example:
     "/export/home/sblqa1/2022_06/ses/applicationcontainer_internal/webapps"
     ```
3. Open the 'ociconfig.properties' file in the internal Tomcat's webapps folder and update the following parameter values as required:

   ```
   user=<CHANGE_ME>
    fingerprint=<CHANGE_ME>
    tenancy=<CHANGE_ME>
    region=<CHANGE_ME>
    compartmentId=<CHANGE_ME>
    key_file=<CHANGE_ME>
   ```

   The value of the key_file should be the location of the .pem file. It is recommended that you keep the .pem file in the same location as the 'ociconfig.properties' file.
4. Verify that the 'ociconfig.properties' file includes the location for the *private key(.pem)* file and ensure that the path is valid.
5. Open the 'ociconfig.properties' file and check the [GENAI] section. If any LLM needs to be changed, please update the same. Similarly, please check the [SPEECH] section. If any Language needs to be added for either Oracle or Whisper model, please add the same. The [GENAI] and [SPEECH] sections will look like below:

   ```
   [GENAI]
   ```

```
chat_model=openai.gpt-4.1
vision_model=openai.gpt-4.1
embed_model=openai.text-embedding-3-small
llama_chat_model=meta.llama-3.3-70b-instruct
llama_vision_model=meta.llama-4-maverick-17b-128e-instruct-fp8
cohere_chat_model=cohere.command-a-03-2025
cohere_embed_model=cohere.embed-multilingual-v3.0
openai_chat_model=openai.gpt-4.1
openai_vision_model=openai.gpt-4.1
openai_embed_model=openai.text-embedding-3-small
xai_chat_model=xai.grok-4
xai_vision_model=xai.grok-4
google_chat_model=google.gemini-2.5-flash
google_vision_model=google.gemini-2.5-flash
custom_endpoint_id=<CHANGE_ME>
cohere_custom_endpoint_id=<CHANGE_ME>
llama_custom_endpoint_id=<CHANGE_ME>
genai_endpoint = https://inference.generativeai.us-chicago-1.oci.oraclecloud.com
genai_region = US_CHICAGO_1
multimodal_image_type = png,jpg,jpeg
multimodal_max_image_size = 5242880
multimodal_image_for_summarize = true
```

6. Import the necessary OCI digital certificate files to siebel trustore and keystore as required – example commands for importing two OCI certificates follow.

   Import the certificate files (digcert1.cer and digicert2.cer) from the command prompt as follows:

   - On Windows:

     ```
     <Java JDK path>\bin>keytool -importcert -keystore <siebel home path>
     \ses\applicationcontainer_internal\siebelcerts\siebeltruststore.jks -storepass siebel
     -file <certificate path>\digicert1.cer -alias "oci-digi01"

     <Java JDK path>\bin>keytool -importcert -keystore <siebel home path>
     \ses\applicationcontainer_internal\siebelcerts\siebeltruststore.jks -storepass siebel
     -file <certificate path>\digicert2.cer -alias "oci-digi02"
     ```

   - On Unix:

     ```
     <Java JDK path>/bin>keytool -importcert -keystore <siebel home path>
     /ses/applicationcontainer_internal/siebelcerts/siebeltruststore.jks -storepass siebel
     -file <certificate path>/digicert1.cer -alias "oci-digi01"

     <Java JDK path>/bin>keytool -importcert -keystore <siebel home path>
     /ses/applicationcontainer_internal/siebelcerts/siebeltruststore.jks -storepass siebel
     -file <certificate path>/digicert2.cer -alias "oci-digi02"
     ```

7. Restart Tomcat.

Uptake of AI/ML powered features with OCI Java SDK 3.54.0

Complete the steps in the following procedure for the uptake of AI/ML powered features with OCI Java SDK 3.54.0.

1. Navigate to **Siebel Internal Tomcat** in the Siebel build and go to the **webapps** folder as mentioned in the below path to find the `oci-java-sdk.zip` file:

   ```
   <Siebel Internal Tomcat's installedLocation>/ses
   /applicationcontainer_internal/webapps.
   ```

   for e.g. in Linux: `/scratch/home/sblqa1/2023_08C004/ses/applicationcontainer_internal/webapps`

for e.g. in Windows: `C:\2023_08C004\ses\applicationcontainer_internal\webapps`

2. The OCI Java SDK JARS zip `oci-java-sdk.zip` will be present in the above location. Please unzip the file to find the JAR files.
3. Copy the JAR files to **lib** folder in the deployed Siebel WAR. Please refer the path as mentioned:

   ```
   <Siebel Internal Tomcat's installed
   Location>/ses/applicationcontainer_internal
   /webapps/siebel/WEB-INF/lib
   ```

   for e.g. in Linux:

   ```
   /scratch/home/sblqa1/2023_08C004/ses/applicationcontainer_internal/webapps/siebel/WEB-INF/lib
   ```

   for e.g. in Windows:

   ```
   C:\2023_08C004\ses\applicationcontainer_internal\webapps\siebel\WEB-INF\lib
   ```

4. Increase the Tomcat JVM Heap size as (and if) required. It can be set to 4GB (Min/Max) or to higher/lower values depending on the build configuration/shape.

   In Linux, the JVM Heap can be set with the help of JAVA_OPTS which can be defined in `setenv.sh` as mentioned, `JAVA_OPTS = %JAVA_OPTS% -Xms4096m -Xmx4096m`

   In Windows, the `tomcat9w.exe` can be run from Tomcat's bin folder to set the values adequately. Please follow these steps:

   - Navigate to the bin folder of Tomcat's installed location as mentioned, `<Siebel Internal Tomcat's installed Location>\bin`
   - Run the exe from this location as below to open a GUI popup, `> tomcat9w.exe //ES//<Tomcat Service Name>`
   - Select the **Java** tab and change the JVM Heap size Min and Max values in Initial memory Pool and Maximum Memory Pool field values respectively.
   - Save the GUI.
   - In the java tab, Java Options end the below command in the end - `Djdk.jar.maxSignatureFileSize=16000000`.
   - Save and close the tab.
5. Restart the Tomcat.

Configure Seed Data and Business Service Access

Make sure that the following Business Services (and their responsibilities) have been added to Siebel CRM and grant the necessary Business Services access to them as shown in the following procedure:

- **SiebelOCIAIWebService,** which is required to enable the Siebel REST API framework for the AI service.
- **SiebelOCIAIService,** which is required to enable the predefined Siebel Application UI use cases.

1. Navigate to the Administration - Application screen, then the Business Service Access view.
2. Create a new record, add the SiebelOCIAIWebService business service and save the record.
3. Create another new record, add the SiebelOCIAIService business service and save the record.
4. In the (Access By Responsibility) Responsibilities applet, add the Siebel Admin responsibility for each new record (created in step 2 and step 3).

5. In the Business Service Method applet, add the respective methods for each new record (created in step 2 and step 3). For example, add the InvokeOCIAI method.
6. Save the records, clear the cache and log out of the application (and back in again) for the changes to take effect.

This table describes the SiebelOCIWebService and SiebelOCIAIService business services in more detail:

Name	Display Name	Class	Business Service Method	Business Service Method Display Name
SiebelOCIAIWebService	SBL_OCI_AI_JAVA_SERVICE	CSSJavaBusinessService	InvokeOCIAI	SBL_OCI_AI_INOVKE
SiebelOCIAIService	SBL_OCI_AI_SERVICE	CSSOCIAIService*	SBL_OCI_AI_INOVKE	SBL_OCI_AI_INOVKE
CSSOCIAIService	ClientAppCommonSVDII	Service	CSSWEUIService	

The SiebelOCIAIService Business Services uses the class described in the following table.

Name	DLL	Object Type	Super Class
CSSOCIAIService	ClientAppCommonSVDII	Service	CSSWEUIService

Configure the Oracle Cloud Infrastructure (OCI) Language AI Service

Complete the steps here to configure the Oracle Cloud Infrastructure (OCI) Language AI service – so that PII will be detected.

1. Modify the relevant user properties in the 'Service Request' Business Component (such as the Business Component field) as required.

   This table describes the user properties to modify and their values:

BC User Property Name	BC User Property Value
BatchDetectLanguagePiiEntities_ Param_1	text:FIELD=Description
BatchDetectLanguagePiiEntities_ Param_2	output:FIELD=PII Flag
BatchDetectLanguagePiiEntities_ Param_3	mode:MASK [Supported options:MASK, REPLACE & REMOVE]

BC User Property Name	BC User Property Value
BatchDetectLanguagePiiEntities_Param_4	isUnmaskedFromEnd:true
BatchDetectLanguagePiiEntities_Param_5	leaveCharactersUnmasked:3
BatchDetectLanguagePiiEntities_Param_6	maskingCharacter:*
BatchDetectLanguagePiiEntities_Param_7	modifyText:false
BatchDetectLanguagePiiEntities_Param_8	domain:ALL (or PII/PHI)
BatchDetectLanguagePiiEntities_Param_9	documentType: <CHANGE_ME> (value like EHR)
BatchDetectLanguagePiiEntities_Param_10	speciality: <CHANGE_ME> (value like paediatrics)
BatchDetectLanguagePiiEntities_Param_11	compartmentId: <CHANGE_ME>

2. For REPLACE masking option, additional Business User properties need to be added as follows:

BC User Property Name	BC User Property Value
BatchDetectLanguagePiiEntities_Param_12	replaceWith:#Not to be Disclosed

3. Other properties that need to be specified when configuring the OCI Language AI service for Text Classification are listed in this table:

BC User Property Name	BC User Property Value
BatchDetectLanguageTextClassification_Param_1	text:FIELD=Description (or any other Field name which has the text content to be classified - Field name should be taken from Applet →List →List Column Value)
BatchDetectLanguageTextClassification_Param_2	endpointId:<CHANGE_ME> (OCID of the custom ML Language model trained on customer data)
BatchDetectLanguageTextClassification_Param_3	compartmentId:<CHANGE_ME>
BatchDetectLanguageTextClassification_Param_4	output:FIELD=Area,Sub-Area (or any other Field names where classified label to be populated)

BC User Property Name	BC User Property Value
BatchDetectLanguageTextClassification_Param_5	modelDelimiter:/ (or any other Delimiter to be used to separate output Fields in the custom data during Model

Configure the Oracle Cloud Infrastructure (OCI) Speech AI Service

Complete the steps here to configure the Oracle Cloud Infrastructure (OCI) Speech AI service.

1. Whisper will be the default model used by Siebel for Speech-to-text transcription. To change the model to Oracle, the 'modelType' parameter in the Business Component User Property can be added with value as 'ORACLE'.
2. Modify the relevant user properties in the 'Service Request Attachment' Business Component as required – such as the Input and Output Bucket and Prefix locations to upload the input (audio) and output (text) file.

   This table lists the user properties to modify.

BC User Property Name	BC User Property Value
CreateTranscriptionServices_Param_1	inputFileName:FIELD=ActivityFileName
CreateTranscriptionServices_Param_2	inputFileExt:FIELD=ActivityFileExt
CreateTranscriptionServices_Param_3	namespace:<CHANGE_ME>
CreateTranscriptionServices_Param_4	compartmentId:<CHANGE_ME>
CreateTranscriptionServices_Param_5	inputBucket:<CHANGE_ME>
CreateTranscriptionServices_Param_6	outputBucket:<CHANGE_ME>
CreateTranscriptionServices_Param_7	inputPrefix:<CHANGE_ME>
CreateTranscriptionServices_Param_8	outputPrefix:<CHANGE_ME>
CreateTranscriptionServices_Param_9	inputFilePath:<CHANGE_ME>
CreateTranscriptionServices_Param_10	outputFilePath:<CHANGE_ME>
CreateTranscriptionServices_Param_11	jobDisplayName:<CHANGE_ME>
CreateTranscriptionServices_Param_12	jobDescription:<CHANGE_ME>
CreateTranscriptionServices_Param_13	jobMaxRetries:<CHANGE_ME>
CreateTranscriptionServices_Param_14	targetLanguageCode:<CHANGE_ME> //not seeded by default

3. Other properties that need to be specified (such as namespace, compartmentId, and job description) when configuring the OCI Speech AI service are listed in the following table.

User Property	Value
NameSpace	OCI Namespace of the User/Administrator.
CompartmentId	Root Compartment of the OCI bucket.
InputBucket	The Bucket in OCI to where the Audio File will be uploaded.
InputPrefix	Folder (or Folder hierarchy) inside the InputBucket in OCI under which the Audio File is stored.
OutputBucket	The Bucket in OCI where the Transcribed Text File will be stored.
OutputPrefix	Folder (or Folder hierarchy) inside the OutputBucket in OCI under which the Text File will be stored.
JobDisplayName	Name of the Transcription Job as desired by the User/Administrator. If no value is supplied, OCI will have a default Job name
JobDescription	Description of the Transcription Job. If no value is supplied, then the field remains as empty in OCI.
JobMaxRetries	The default value is 4. For larger speech files, change the value to a higher number as required.
InputFileName:FIELD	The Business Component Field which is mapped to the Input File Name that needs to be transcribed.
InputFileExt:FIELD	The Business Component Field which is mapped to the File Extension format of the Input File.
InputFilePath	Input path for the file which needs to be transcribed from OCI Bucket<optional user property which is considered only in case of New URL>.
OutputFilePath	Output Path of the transcribed file <optional user property which is considered only in case of New URL>.

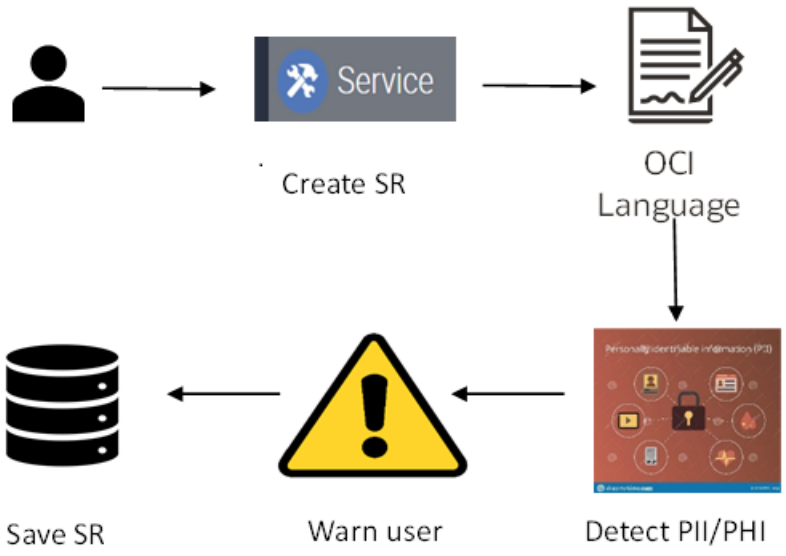
Artificial Intelligence (AI) Services Use Cases

This topic describes the following sample Artificial Intelligence (AI) use cases:

- *Use Case 1: Identify and Flag PII/PHI in Service Request Description*
- *Use Case 2: Transcribe Customer Calls and Attach Text File to Service Request*

Use Case 1: Identify and Flag PII/PHI in Service Request Description

In this scenario (illustrated in the following image), the OCI Language AI service checks whether the service request (SR) description field contains any PII/PHI, and if so, warns the user before saving the SR to the system.



As shown in this image, a typical flow involved in using the OCI Language AI service to identify and flag PII/PHI in the service request (SR) description is as follows:

1. A call center agent creates an SR, for example, where the Description field contains the following PII information:

```
The customers MEDICAL RECORD NUMBER is MRN:56453241-A and can be reached at 9989898989 or
 charles@gmail.com.
```

2. When the SR is saved:
    a. Siebel invokes the OCI Language AI service to check the Description field to see if it contains any PII/PHI information.
    b. If the Descriptions field contains PII/PHI information, then Siebel warns the user before saving the SR with a message similar to the following:

```
Personally Identifiable Information
TELEPHONE_NUMBER: 9989898989
EMAIL: charles@gmail.com
MEDICAL_RECORD_NUMBER: MRN:56453241-A

OK
```

    c. When the user clicks OK to this message, the PII Flag field value for the SR changes to Y.

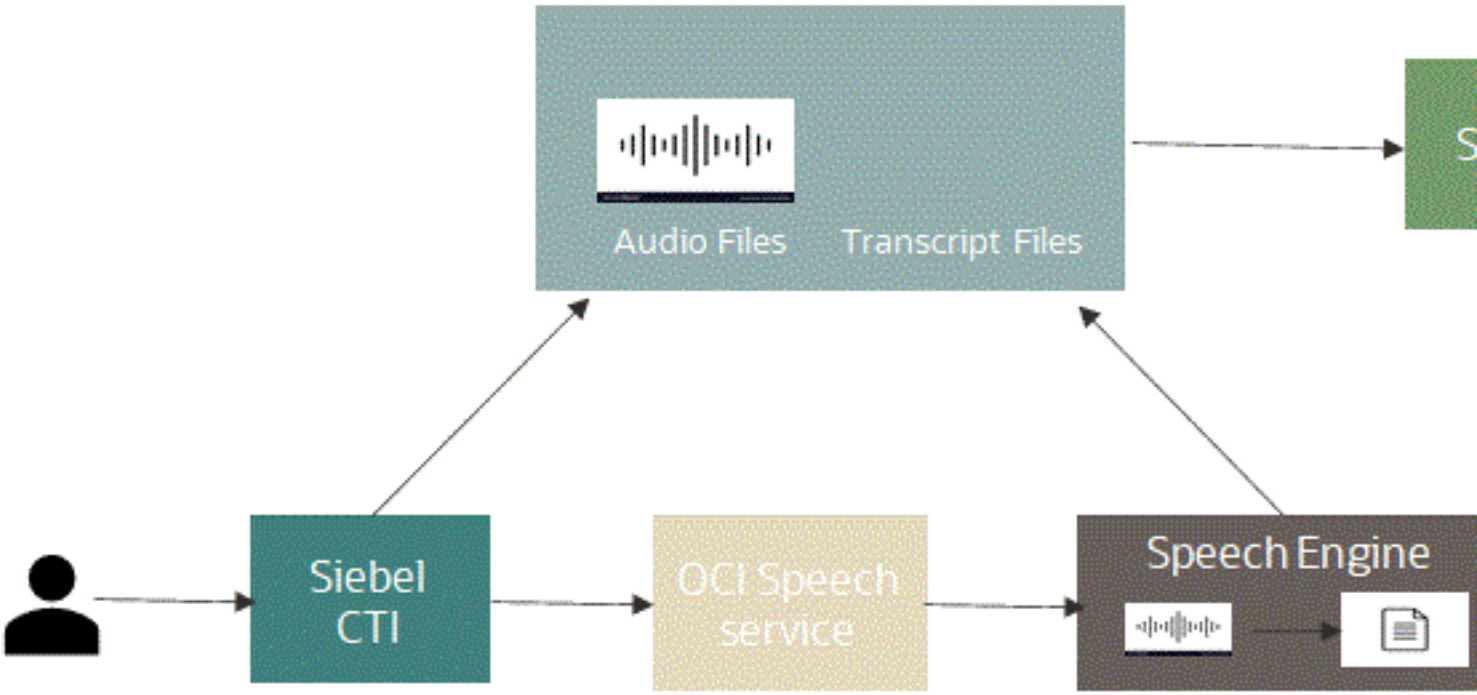
You can extend this functionality to any Siebel business component. Currently, only one field per business component is supported to extract PII from input text, for example, as follows:

- The 'Description' field in the 'Service Request' business component.
- The 'Comment' field in the 'Quote' business component.

However, administrators can configure a list of PII/PHI to be identified in the configured field.

Use Case 2: Transcribe Customer Calls and Attach Text File to Service Request

In this scenario (illustrated in the following image), the OCI Speech AI service automatically converts file-based audio data into highly accurate transcriptions.



As shown in this image, a typical flow involved in using the OCI Speech AI service to transcribe customer calls and attach the text file to the service request (SR) is a follows:

1. A call center agent records a conversation with a customer.
2. The call center agent attaches the conversation (audio file) to an SR.

   You can use either the New File or New URL button to attach an audio file to an SR and (in both cases) you must specify the entire path to the audio file. The audio file can be in the following formats - WAV, AAC, AC3, AMR, AU, FLAC, M4A, MKV, MP3, MP4, OGA, OGG, WEBM. For example:

   - Click New File to add a wav file (oracle-advertising.wav) from your local machine.
   - Click New URL to add the full URL path to the speech file (in .wav format) from the OCI object storage: `https://objectstorage.ap-mumbai-1.oraclecloud.com/n/siebeldev/b/priyabucket-20220414-1900/o/oracle-advertising.wav`.
3. The call center agent clicks Transcribe.

   Doing this invokes the OCI Speech AI service which transcribes the audio in the wav file to text (in a txt file).
4. The txt file, containing the transcribed audio from the wav file, is attached to the SR.

   Click the Attachment Name field to download the txt file, .

You can extend this functionality to any Siebel business component. Currently, only one audio file can be transcribed at a time.

Siebel REST APIs for Artificial Intelligence (AI) Services

The Oracle Cloud Infrastructure (OCI) Language and Speech AI Services can be called via Siebel REST APIs:

- The Language API detects Personal Identifiable Information (PII) from single or multiple text inputs.
- The Speech API transcribes speech/audio content to text and allows you to upload/download files between OCI Object storage and the Siebel Server local file system.

Siebel REST API can be used by Siebel internal and external clients to invoke OCI AI Services, for example as follows (for various use cases), where SiebelOCIAIWebService is the Web Service name and InvokeOCIAI is the Web Service method:
`https://<hostname:port>/siebel/v1.0/service/SiebelOCIAIWebService/InvokeOCIAI`

The API Request payload must contain the 'Context' parameter and the OCI AI Domain followed by the specific functionality in that domain. For example, use the `Speech:CreateTranscriptionServices` context parameter to transcribe AI speech to text.

The context parameters for Language are as follows:

- `Language:BatchDetectLanguageEntities`
- `Language:BatchDetectDominantLanguage`
- `Language: BatchDetectLanguagePiiEntities`
- `Language:BatchDetectLanguageSentiments`
- `Language:BatchDetectLanguageTextClassification`
- `Language:BatchLanguageTranslation`

For more information about using the Siebel REST APIs for AI Services, see the following topics and the *Siebel REST API Guide*.

- *Why Keep an Oracle Learning Offering Overbooked*
- *Example of Adding a Card Component to an Existing Canadian Involuntary Deduction Card*
- *Setup Tasks for Implementing Payroll for Saudi Arabia*

Siebel REST API for the Language AI Service

The REST API for Language AI service supports named entity recognition (NER) with an option to select custom PII. Note the following about using this API:

1. You can specify to detect one or more named entities in the text.
2. You can specify to detect PII entities only by mentioning them in the request payload.
3. Input text (including batch loads) is analyzed for named entities.
4. The response is filtered based on the PII list specified.

   If no PII is specified, then all named entities will be recognised in the response.

5. The request parameters to use when calling the Language NER API are described in the following table.

Request Parameter	Description
key	This parameter refers to the document index (SR Id or other identifier). The value is in the format **doc_<index>** where **<index>** is an integer. Some example values are doc1, doc2, and so on.
text	The input text from which the named entities are to be recognised.
PII	The subset of named entities – which you can select from the entire list of Named Entities.

6. The output parameters for the Language NER API service are described in the following table.

Output Parameter	Description
Status	The status of the operation.
OPCRequestId	The request/transaction ID maintained by the OCI. This identifier is also used by any debug or DevOps operation.
key	Refers to the document index using the SR Id or other identifier.
Entities	An array of named entities identified by OCI in the text. Each array element has a 'type' and 'text' element where: o **type** refers to the type of named entity (such as PHONE_NUMBER, EMAIL, ORG, EVENT, and so on). o **text** refers to the exact text (word/number) representing the entity.

Sample API to be Invoked (POST)

```
https://<hostname:port>/<siebel>/<v1.0>/service/SiebelOCIAIWebService/InvokeOCIAI
```

Sample Request Payload

```
{

"body" :
 {
 "Context": "Language:BatchDetectLanguageEntities",
 "BatchDetectLanguageEntities_Param_1":
 {
 "documents":[
 {
 "key": "doc1",
 "text":"Oracle has partnered with zoom for the best audio experience. Please select computer audio when
 joining the meeting. You can download the Zoom mobile app on iOS or Android for HD audio. Zoom interface is
```

```
very simple and intuitive to use. For educational vieos on other amazing Oracle Products, please reach out
to edus_in@oracle.com"
},
{
"key": "doc2",
"text": "Red Bull Racing Honda is a car company."
} ,
{
"key":"doc3",
"text": "In 2020 people worldwide moved to working remotely because of the COVID-19 pandemic. As a
result, collaborative tools like video conferencing, email and chat have become critical, as they allow
employees to perform their jobs from home. However, the human element of work has been much more difficult
for workforces to navigate, bringing mental health and burnout challenges to the forefront. To support
current remote workforce challenges, organisations are turning to emerging technologies such as artificial
intelligence (AI), machine learning (ML), and chatbots in partnership with Workforce Intelligence to
benefit million of users."
} ,
{
"key":"doc4",
"text": "OCI recently added new services to existing compliance program including SOC, HIPAA, and ISO to
enable our customers to solve their use cases. We also released new white papers and guidance documents
related to Object Storage, the Australian Prudential Regulation Authority (APRA), and the Central Bank
of Brazil. These resources help regulated customers better understand how OCI supports their regional and
industry-specific compliance requirements. Not only are we expanding our number of compliance offerings and
regulatory alignments, we continue to add regions and services at a faster clip."
}
]
},
"BatchDetectLanguageEntities_Param_2":"pii:email,phone_number,org"
}
}
```

Other entity types (such as URL, IPADDRESS, and so on) can be specified for example as follows:

```
"Context": "Language:BatchDetectLanguageEntities",
"BatchDetectLanguageEntities_Param_2":"pii:email,phone_number,org,person,URL,IPADDRESS"
```

The following will detect and return all PII entity information in the given text documents:

```
"Context": "Language:BatchDetectLanguageEntities",
"BatchDetectLanguageEntities_Param_3":"PII"
```

Sample Corresponding Response Payload

```
{
 "Status": "Success",
 "OPCRequestId": "850E5831E5BE437FA6DE27EE61583F77/2D62DEA891C61E675399E2F20BACC56F/
E749D506C91009EE9B90BF8E7714E7B7",
 "documents": {
 "Response": [
 {
 "key": "doc4",
 "entities": [
 {
 "type": "ORG",
 "length": "3",
 "offset": "0",
 "text": "OCI",
 "subType": "null",
 "score": "1.0"
 },
 {
 "type": "ORG",
 "length": "3",
 "offset": "73",
```

```
"text": "SOC",
"subType": "null",
"score": "1.0"
},
{
"type": "ORG",
"length": "5",
"offset": "78",
"text": "HIPAA",
"subType": "null",
"score": "1.0"
},
{
"type": "ORG",
"length": "3",
"offset": "89",
"text": "ISO",
"subType": "null",
"score": "1.0"
},
{
"type": "ORG",
"length": "42",
"offset": "231",
"text": "Australian Prudential Regulation Authority",
"subType": "null",
"score": "1.0"
},
{
"type": "ORG",
"length": "12",
"offset": "290",
"text": "Central Bank",
"subType": "null",
"score": "0.9999995827674866"
},
{
"type": "ORG",
"length": "3",
"offset": "377",
"text": "OCI",
"subType": "null",
"score": "1.0"
}
]
},
{
"key": "doc3",
"entities": {
"type": "ORG",
"length": "22",
"offset": "586",
"text": "Workforce Intelligence",
"subType": "null",
"score": "1.0"
}
},
{
"key": "doc2",
"entities": {
"type": "ORG",
"length": "21",
"offset": "0",
"text": "Red Bull Racing Honda",
"subType": "null",
"score": "1.0"
```

```
 }
 },
 {
 "key": "doc1",
 "entities": [
 {
 "type": "ORG",
 "length": "6",
 "offset": "0",
 "text": "Oracle",
 "subType": "null",
 "score": "1.0"
 },
 {
 "type": "ORG",
 "length": "6",
 "offset": "277",
 "text": "Oracle",
 "subType": "null",
 "score": "1.0"
 },
 {
 "type": "EMAIL",
 "length": "18",
 "offset": "315",
 "text": "edus_in@oracle.com",
 "subType": "null",
 "score": "0.9752325245161998"
 }
 ]
 }
 ]
 }
}
```

Sample API to be Invoked (POST)

```
https://<siebel server or host>:<host port>/siebel/v1.0/service/SiebelOCIAIWebService/InvokeOCIAI
```

Example:

```
https://phoenix201061.appsdev1.fusionappsdphx1.oraclevcn.com:16690/siebel/v1.0/service/
SiebelOCIAIWebService/InvokeOCIAI
```

authentication : `sadmin/ldap`

Sample Payload for Sentiment Analysis

```
//Request payload:
{
 "body": {
 "Context": "Language:BatchDetectLanguageSentiments",
 "BatchDetectLanguageSentiments_Param_1": {
 "documents": [
 {
 "key": "doc1",
 "text": "Oracle has partnered with zoom for the best audio experience. Please select computer audio when
 joining the meeting. You can download the Zoom mobile app on iOS or Android for HD audio. Zoom interface is
 very simple and intuitive to use. For educational vieos on other amazing Oracle Products, please reach out
 to edus_in@oracle.com"
 }
 ]
 }
 }
```

```
}
```

Sample Corresponding Response Payload

```
{
 "Status": "Success",
 "OPCRequestId":
 "E20E0932B4054EACBE6BB33F0A3CC146/43994C43A1BED9FA5C8AFA622AE952E5/5B6D9C80A0325E6DF47BD7A871756495",
 "documents": {
 "Response": {
 "key": "doc1",
 "documentSentiment": "Positive",
 "Sentiments": {
 "positive": "0.97745436",
 "negative": "0.0",
 "mixed": "0.022545636",
 "neutral": "0.0"
 }
 }
 }
}
```

Sample Payload for Text Classification

```
//Request payload
{
 "body": {
 "Context": "Language:BatchDetectLanguageTextClassification",
 "BatchDetectLanguageTextClassification_Param_1": {
 "documents": [
 {
 "key": "doc1",
 "text": "Oracle has partnered with zoom for the best audio experience. Please select computer audio when
 joining the meeting. You can download the Zoom mobile app on iOS or Android for HD audio. Zoom interface is
 very simple and intuitive to use. For educational vieos on other amazing Oracle Products, please reach out
 to edus_in@oracle.com"
 },
 {
 "key": "doc2",
 "text": "OCI recently added new services to existing compliance program including SOC, HIPAA, and ISO to
 enable our customers to solve their use cases. We also released new white papers and guidance documents
 related to Object Storage, the Australian Prudential Regulation Authority (APRA), and the Central Bank
 of Brazil. These resources help regulated customers better understand how OCI supports their regional and
 industry-specific compliance requirements. Not only are we expanding our number of compliance offerings and
 regulatory alignments, we continue to add regions and services at a faster clip."
 },
 {
 "key": "doc3",
 "text": "Red Bull Racing Honda is a car company."
 }
 ]
 }
 }

}
```

Sample Corresponding Response Payload

```
{
 "Status": "Success",
 "OPCRequestId":
 "7A0A64D4CD2841808B250320F28A2ECC/89E562C0784B8EA5F48B8FB5FA463FF3/20A50715BA04A46CC7398C9B856F53E8",
 "documents": {
 "Response": [
 {
```

```
 "key": "doc3",
 "TextClassification": {
 "label": "Autos and Vehicles/Motor Vehicles",
 "score": "0.371180564"
 }
 },
 {
 "key": "doc2",
 "TextClassification": {
 "label": "Business and Industry/Business Operations",
 "score": "0.193597227"
 }
 },
 {
 "key": "doc1",
 "TextClassification": {
 "label": "Computer and Electronics/Software",
 "score": "1.0"
 }
 }
 ]
 }
}
```


Sample Payload for Detection of Dominant Language

```
//Request payload
{
 "body": {
 "Context": "Language:BatchDetectDominantLanguage",
 "BatchDetectDominantLanguage_Param_1": {
 "documents": [
 {
 "key": "doc1",
 "text": "Oracle has partnered with zoom for the best audio experience. Please select computer audio when
 joining the meeting. You can download the Zoom mobile app on iOS or Android for HD audio. Zoom interface is
 very simple and intuitive to use. For educational vieos on other amazing Oracle Products, please reach out
 to edus_in@oracle.com"
 },

 {
 "key": "doc3",
 "text": "Buongiorno! Oggi vi presento la mia famiglia. Io sono il padre, mi chiamo Gennaro Pirlo, ho
 trentasette anni, e lavoro come scrittore e giornalista da quando ne avevo venti.Mia moglie si chiama
 Antonella Totti, ha trentacinque anni ed è una splendida attrice di teatro.La nostra famiglia è composta
 anche da altre due persone, i nostri figli, Manuela che ha diciassette anni, e Marco che ha quindici anni,
 e poi c'è anche Tremendo, il cane che vive con noi da nove anni, ed è parte della famiglia. Viviamo tutti
 nella nostra splendida casa con un grande giardino."
 }
 ]
 }
 }

}
```

Sample Corresponding Response Payload


```
{
 "Status": "Success",
 "OPCRequestId": "E3DAADACD8784125AAA7BC38316813E4/1714D759EB3309119E0C4E7ADC9390E0/
F170A38C6C089C7FF7289EA4D3F9897F",
 "documents": {
 "Response": [
 {
 "key": "doc3",
```

```
 "languages": {
 "name": "Italian",
 "code": "it",
 "score": "1.0"
 }
 },

 {
 "key": "doc1",
 "languages": {
 "name": "English",
 "code": "en",
 "score": "1.0"
 }
 }
 ]
 }
}
```

Sample Payload for Language Translation

```
//Request payload
{
 "body": {
 "Context": "Language:BatchLanguageTranslation",
 "BatchLanguageTranslation_Param_1": {
 "documents": [
 {
 "key": "doc1",
 "text": "Oracle has partnered with zoom for the best audio experience. Please select computer audio when
 joining the meeting. You can download the Zoom mobile app on iOS or Android for HD audio. Zoom interface is
 very simple and intuitive to use. For educational vieos on other amazing Oracle Products, please reach out
 to edus_in@oracle.com"
 }
 ]
 },
 "BatchLanguageTranslation_Param_2": "TargetLanguageCode:it"
 }

}
```

Sample Corresponding Response Payload

```
{
 "Status": "Success",
 "OPCRequestId":
 "19CFED98A2F140C7B60F264C8D62CEB4/3E01A076AF57008E32532C2CA3934FD9/4544F0D0E7F084C36AA821B6565102E3",
 "documents": {
 "Response": {
 "key": "doc1",
 "LanguageTranslation": {
 "translatedText": "Oracle ha stretto una partnership con zoom per offrire la migliore esperienza audio.
 Selezionare l'audio del computer quando si partecipa alla riunione. È possibile scaricare l'app mobile Zoom
 su iOS o Android per l'audio HD. L'interfaccia Zoom è molto semplice e intuitiva da usare. Per conoscere
 gli eventi formativi su altri straordinari prodotti Oracle, contatta edus_in@oracle.com"
 }
 }
 }
}
```

Sample Payload for LanguagePiiEntities

```
//Request Payload
{
 "body": {
```

```
 "Context": "Language:BatchDetectLanguagePiiEntities",
 "BatchDetectLanguagePiiEntities_Param_1": {
 "documents": [
 {
 "key": "doc1",
 "text": "A SWIFT (Society for Worldwide Interbank Financial Telecommunication) code, also known as a BIC
 (Bank Identifier Code) number is HDFC000009"
 }
 ]
 },
 "BatchDetectLanguagePiiEntities_Param_2": "mode:MASK",
 "BatchDetectLanguagePiiEntities_Param_3": "isUnmaskedFromEnd:true",
 "BatchDetectLanguagePiiEntities_Param_4": "leaveCharactersUnmasked:3",
 "BatchDetectLanguagePiiEntities_Param_5": "maskingCharacter:*"
 }
}
```

Sample Corresponding Response Payload

```
{
 "Status": "Success",
 "OPCRequestId": "3E866FFD9FDB4F92B2093DD55CA1713D/
E7A63B05AB6FF208BE87332C70321DE5/4A1C05FCCDE9A3F23FD13B14CBDBBB4A",
 "documents": {
 "Response": {
 "key": "doc1",
 "entities": {
 "type": "BANK_SWIFT",
 "length": "10",
 "maskedText": "A SWIFT (Society for Worldwide Interbank Financial Telecommunication) code, also known as a
 BIC (Bank Identifier Code) number is *******009",
 "languageCode": "en",
 "offset": "129",
 "text": "HDFC000009",
 "score": "0.9926007986068726"
 }
 }
 }
```

For more information about using the Siebel REST API for Language AI Service, see Siebel REST API Guide.

Siebel REST API for the Speech AI Service

The following describes how the REST API for Speech AI service is used:

1. You call the REST API for Speech AI service via the Speech CreateTranscription API.

   This API allows you to select an audio file present in the Server's local file system. The other request parameters to use when calling the Speech CreateTranscription API are described in the following table.

Request Parameter	Description
InputFile	The Audio File to be transcribed to text.
InputFilePath	The Audio File path in the Server's local file system. The path should be accessible by the Siebel Server and local to its file system.

Request Parameter	Description
NameSpace	OCI Namespace of the User/Administrator.
CompartmentId	Root Compartment of the OCI bucket.
InputBucket	The Bucket where the Audio File will be uploaded. If the InputFilepath is empty, then the API will directly take the Audio file from this bucket.
InputPrefix	Folder (or Folder hierarchy) inside the InputBucket where the Audio File is present.
JobDisplayName:	Name of the Transcription Job as desired by the User/Administrator. If no value is specified, OCI will use a default job name.
JobDescription	Description of the Transcription Job. If no value is specified, then the field remains empty in OCI.
OutputBucket	The Bucket where the Transcribed Text File will be stored.
OutputPrefix	Folder (or Folder hierarchy) inside the OutputBucket where the Text File is stored.
OutputFile	The transcribed Text File name.
OutputFilePath	The transcribed Text File path in the Server's local file system. Siebel Server must be able to access this location.
JobHeartbeatTimer	The heartbeat interval to check for the Transcription Job. The default value is 10000 milliseconds (=10 seconds). Do not add this parameter to the request payload; allow the default value to persist. In the case of large or small audio files, change the default value by adding the parameter to the request payload
JobMaxRetries	The maximum retries to check for the status of the Transcription Job from OCI. The default value is 4. Do not add this parameter to the request payload; allow the default value to persist. In the case of large or small audio files,change the default value by adding the parameter to the request payload. In the case where MaxRetries is exhausted and the Transcription Job has not yet finished (JobStatus is 'In Progress'), the API returns adequately with the Job Id and the necessary details to query the Job further.

2. After the Speech CreateTranscription API completes authentication, the audio file is uploaded to the OCI bucket.

3. A Transcription Job is then created which takes the audio file from the InputBucket and converts it to a text file.

   If the job succeeds straightaway, then the transcribed text file is stored in the OutputBucket. The API will further query this location, get the JSON file and download it to the Siebel Server local file system in either JSON or text format (for end user consumption).

   If the job takes longer to complete, then the API returns a Job Status of In Progress in the Response payload. The client will then have to fetch the JobStatus and download the transcribed file once the job completes. These additional actions require the client to call the following API:

   - DownloadUtility – for more information, see *Utility APIs and Methods Supporting the Speech AI Service*.

4. The output parameters for the Speech CreateTranscription API are described in the following table.

Request Parameter	Description
Status	Status of the operation.
OutputBucket	The bucket where the Transcribed Text File is stored.
OutputPrefix	The folder name (and path) where the Transcribed File is present in the OCI bucket.
ObjectLocation	The Transcribed File name along with the complete folder path (or folder hierarchy) under the OutputBucket.
OutputPath	The Transcribed File (in JSON/Text format) with complete path to the Server local file system where it has been downloaded from the OCI bucket.
NameSpace	OCI Namespace of the User/Administrator.
OCID	OCID of the User/Administrator.
OPCRequestId	The Request/transaction ID maintained by OCI. This identifier will be used for any debug or DevOps operation.

Sample API to be Invoked (POST)

```
https://<hostname:port>/<siebel>/<v1.0>/service/SiebelOCIAIWebService/InvokeOCIAI
```

Sample Request Payload on Windows

```
"body" :
 {
 "Context": "Speech:CreateTranscriptionServices",
 "CreateTranscriptionServices_Param_1" : "namespace:siebeldev",
 "CreateTranscriptionServices_Param_2" : "inputBucket:bucket-20220414-1900",
 "CreateTranscriptionServices_Param_3" : "outputBucket:bucket-20220414-1900",
 "CreateTranscriptionServices_Param_4" : "jobDisplayName:DEMOJob",
 "CreateTranscriptionServices_Param_5" : "outputPrefix:DEMOPrefix",
```

```
 "CreateTranscriptionServices_Param_6" : "compartmentId:ocid1.tenancy.oc1..XXXXX",
 "CreateTranscriptionServices_Param_7" : "inputFile:oracle-advertising.wav",
 "CreateTranscriptionServices_Param_8" : "jobDescription:Invoke OCI Speech",
 "CreateTranscriptionServices_Param_9" : "inputFilePath\\Users\\2024_10C002\\AIMLfiles",
 "CreateTranscriptionServices_Param_10" : "inputPrefix:Demo",
 "CreateTranscriptionServices_Param_11" : "outputFilePath: C:\\Users\\2024_10C002\\AIMLfiles ",
 "CreateTranscriptionServices_Param_12" : "outputFile:oracle-advertisingOracleWAV2.json",
 "CreateTranscriptionServices_Param_13" : "jobHeartbeatTimer:100000",
 "CreateTranscriptionServices_Param_14" : "jobMaxRetries:30",
 "CreateTranscriptionServices_Param_15" : "inputFileName:FIELD=ActivityFileName",
 "CreateTranscriptionServices_Param_16" : "inputFileExt:FIELD=ActivityFileExt",
 "CreateTranscriptionServices_Param_17" : "targetLanguageCode:ENU"
 }
```

Sample Corresponding Response Payload on Windows

```
{
 "OutputPath": C:\\Users\\2024_10C002\\AIMLfiles\\oracle-advertisingOracleWAV2.json",
 "ObjectLocation": "DEMOPrefix/job-aXXt3alfa/siebeldev_bucket-20220414-1900/oracle-advertising.wav.json",
 "Namespace": "siebeldev",
 "OutputBucket": "bucket-20220414-1900",
 "OCID": "ocid1.aispeechtranscriptionjob.oc1.apmumbai1.amaXXXXyixjjrmrt3alfa",
 "OPCRequestId": "bom-1:NMuIKvBVFOIWyAVGASsRU5XXXXXXXXX7vkF8Ld_b0GO",
 "Status": "success",
 "JobStatus": "Succeeded",
 "OutputPrefix": "DEMOPrefix/job-amaaaaaa4n2rr5iarea72dfoh75XXXXXXXmrt3alfa/"
}
```

Sample Request Payload on Unix

```
"body" :
 {
 "Context": "Speech:CreateTranscriptionServices",
 "CreateTranscriptionServices_Param_1" : "namespace:siebeldev",
 "CreateTranscriptionServices_Param_2" : "inputBucket:bucket-20220414-1900",
 "CreateTranscriptionServices_Param_3" : "outputBucket:bucket-20220414-1900",
 "CreateTranscriptionServices_Param_4" : "jobDisplayName:DEMOJob",
 "CreateTranscriptionServices_Param_5" : "outputPrefix:DEMOPrefix",
 "CreateTranscriptionServices_Param_6" : "compartmentId:ocid1.tenancy.oc1..XXXXX",
 "CreateTranscriptionServices_Param_7" : "inputFile:oracle-advertising.wav",
 "CreateTranscriptionServices_Param_8" : "jobDescription:Invoke OCI Speech",
 "CreateTranscriptionServices_Param_9" : "inputFilePath:/scratch/home/sblqa1/2024_10C002/AIMLfiles",
 "CreateTranscriptionServices_Param_10" : "inputPrefix:Demo",
 "CreateTranscriptionServices_Param_11" : "outputFilePath:/scratch/home/sblqa1/2024_10C002/AIMLfiles",
 "CreateTranscriptionServices_Param_12" : "outputFile:oracle-advertisingOracleWAV2.json",
 "CreateTranscriptionServices_Param_13" : "jobHeartbeatTimer:100000",
 "CreateTranscriptionServices_Param_14" : "jobMaxRetries:30",
 "CreateTranscriptionServices_Param_15" : "inputFileName:FIELD=ActivityFileName",
 "CreateTranscriptionServices_Param_16" : "inputFileExt:FIELD=ActivityFileExt",
 "CreateTranscriptionServices_Param_17" : "targetLanguageCode:ENU"
}
```

Sample Corresponding Response Payload on Unix

```
{
"OutputPath": "/scratch/home/sblqa1/2024_10C002/AIMLfiles/oracle-advertisingOracleWAV2.json",
 "ObjectLocation": "DEMOPrefix/job-XXXX /siebeldev_bucket-20220414-1900_OnJune15/oracle-
advertising.wav.json",
"Namespace": "siebeldev",
"OutputBucket": "bucket-20220414-1900",
"OCID": "ocid1.aispeechtranscriptionjob.oc1.ap-mumbai-1.amaaaaaaXXXXXXXrt3alfa",
"OPCRequestId": "bom-1:NMuIKvBVFOIWyAVGASsRXXXX_b0GO",
"Status": "success",
"JobStatus": "Succeeded",
"OutputPrefix": "DEMOPrefix/job-amaaaaaa4n2rr5iarea72dfXXXXXlfa/"
```

}

Utility APIs and Methods Supporting the Speech AI Service

A separate utility service is used to upload a file to OCI and to download a file from OCI. For more information, see the following topics:

- *Upload API Utility*
- *Download API Utility*

Upload API Utility

A separate utility service is used to upload a file to OCI. The typical steps involved in this are as follows:

1. You choose to upload the audio file from the Server's local file system to the OCI bucket (InputBucket parameter).
2. You then call the Speech AI service API with an empty InputFilePath.
3. The Speech AI service API takes the audio file directly from the OCI bucket.
4. In the Request Payload, configure the following parameters:
    - **FilePath.** This parameter must specify the source/input audio file location in the Siebel Server local file system.
    - **FileName.** This parameter must specify the audio file name.
    - **Prefix.** This parameter, also known as the target/output location, must specify the OCI folder/prefix name (and path) where the audio file will be uploaded to in the OCI bucket.

Download API Utility

A separate utility service is used to download a file from the OCI bucket to the Siebel Server local file system. The following parameters must be configured in the Request Payload:

- **ObjectLocation.** This parameter must specify the source/OCI location of the object (audio file).
- **OutputFilePath.** This parameter must specify the target location on the Siebel Server local file system for the transcribed Text File.
- **OutputFile.** This parameter must specify the transcribed Text File name (text or json).

Accessing Large Language Models (LLMs) with APIs

The API-based integration methodology enables developers to connect the Siebel CRM application to large language models (LLMs) that support their specific Generative AI use-cases. It provides a single interface to access LLMs from various model providers including Cohere, Google Gemini, OpenAI gpt, xAI Grok and Meta Llma models available on Oracle Cloud Infrastructure (OCI).

For more information, see:

- *About API based Integration*
- *Setting Up and Configuring API Access*

About API based Integration

The API-based integration provides the option to call different Generative AI endpoints The APIs can be invoked using external clients like Postman or from any Siebel interface layer (For Example: Siebel UI).

The API request includes “provider” and “model” as parameters. In addition to this, other configuration parameters that are required vary depending on the model provider and the chosen LLM.

Setting Up and Configuring API Access

This topic explains the following:

Token-based and API Key-based Authentication

To use access token authorization:

1. Provide the credential (Authorization) details in the ociconfig.properties file, according to your respective provider.
2. Update Authorization header Bearer value. Here <token> is just for reference.
3. You can also authorize via api_key. For example: [AUTH] Authorization=api_key <value>
4. Import the necessary certificates if required in your environment.

Provider	ociconfig.properties

OCI	`[AUTH]`
	`user=<oci_user_id>`
	`fingerprint=<oci_user_finger_print>`
	`tenancy=<tenancy_id>`
	`region=<region>`
	`key_file=<.pem file path>`

OpenAI	`[AUTH]` `Authorization=Bearer <token>`

Google	`[AUTH]`
	`Authorization=Bearer <token>`
	`X-goog-user-project=<project_id>`

Invoking Generative AI Use Cases from Siebel UI

To invoke Generative AI use cases from the Siebel UI, create Business Component User Prop as shown below:

Note: User Prop “provider” and “model” are mandatory.

Here's an example for the Sentiment Analysis use case from the Action Business Component for OCI Generative AI endpoints.

BU User Property Name	BC User Property Value
GenerateChatText_Param_1	compartmentId:ocid1.tenancy.oc1..XXXXXXXXXXX
GenerateChatText_Param_2	servingMode:ON_DEMAND
GenerateChatText_Param_3	input:FIELD=Description
GenerateChatText_Param_4	prompt:SentimentPrompt
GenerateChatText_Param_5	output:FIELD=Sentiment
GenerateChatText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL
GenerateChatText_Param_7	outputToOtherBC:BUSCOMP=Service Request,FIELD=Sentiment
GenerateChatText_Param_8*	provider:oci**
GenerateChatText_Param_9*	cohere.command-r-XXXX**

Here's an example for the Sentiment Analysis use case from the Service Request Business Component for sample Model Provider.

BU User Property Name	BC User Property Value
GenerateChatText_Param_1	compartmentId:ocid1.tenancy.oc1..XXXXXXX
GenerateChatText_Param_2	servingMode:ON_DEMAND
GenerateChatText_Param_3	input:FIELD=Description
GenerateChatText_Param_4	prompt:SentimentPrompt
GenerateChatText_Param_5	output:FIELD=Sentiment
GenerateChatText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL
GenerateChatText_Param_7*	provider:openai**
GenerateChatText_Param_8*	model:gpt-4o**

*New parameters to be added (remaining are predefined parameters)

**Specify value based on "provider" and "model"

Passing JSON Parameters

To send JSON parameters in GET and POST requests, define the JSON parameters such as path, action and so on as shown below:

Parameter Name	Definition
Path	Specify the URI published by the LLM provider
Action	Supported POST and GET
requestPayload	Input payload published by the LLM provider need to be enclosed within this parameter
Prompt	Dynamic texts sent by the client (Siebel, Postman, and so on) would be substituted
responseField	Response text parameter name, can differ across the provider + model combinations Example: For Cohere, the response field is “text”, for OpenAI response field is “content” and so on. It's used to display the response in UI

Here's an example is for outbound calls in JSON format for OCI Cohere Command R:

JSON File Name	JSON File Contents
`oci_cohere.command-r-08-2024`	`{ "path": "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com/20231130/actions/chat", "requestPayload": { "chatRequest": { "apiFormat": "COHERE", "message": "<prompt>" }, "compartmentId": "<compartmentId>", "servingMode": { "modelId": "cohere.command-r-08-2024", "servingType": "ON_DEMAND" } }, "action": "POST", "responseField": "text" }`

Here's an example is shown here for outbound calls in JSON format for a sample Model Provider:

JSON File Name	JSON File Contents
`google_gemini-1.5-flash-001.json`	`{ "path": "https://us-central1- aiplatform.googleapis.com/v1/projects/suryageminiproject1/locations/us-central1/publishers/google/models/gemini-1.5-flash-001:generateContent", "action": "POST", "responseField": "text", "requestPayload": { "contents": [{ "role": "user", "parts": [{ "text": "<prompt>" }] }] }}`

Making API Calls using Postman

Here's an example for making API calls using Postman for OCI:

URL	Body

https://<host- name>/Siebel/v1.0/service/SiebelGenericAIWebService/InvokeGenericAI	<pre>{ "body": {
	"provider": "oci",
	"prompt": "What is helidon SE",
	"model": "cohere.command-r-08-2024",
	"compartmentId":"ocid1.tenancy.oc1..**********"
	}
	}</pre>

Here's an example for making API calls using Postman for a sample Model Provider:

URL	Body

OpenAI	<pre>"body": {
	"provider": "openai",
	"prompt": "Tell me the capital of Karnataka",
	"model": "gpt-4o"
	}</pre>

Repository Changes

Here are the repository changes when setting up and configuring API access:

- Create a new Business Service ‘SiebelGenericAIWebService’ and define the Business Service at the Application User Prop.
- Grant ‘Business Service Access’ to the new Business Service ‘SiebelGenericAIWebService’.

Business Service:

Name	Project	Class	Display Name – String Reference
SiebelGenericAIWebService	EAI Business Services	CSSJavaBusinessService	SBL_GENERIC_AI_WEB_SERVICE

Business Service User Prop:

Name	Value
@class	com.siebel.ociaiservice.GenericAIWebService

3 Generative AI

About Oracle Cloud Infrastructure (OCI) Generative AI Service

Oracle Cloud Infrastructure (OCI) Generative AI is a fully managed service that provides a set of state-of-the-art, customizable large language models (LLMs) that cover a wide range of use cases, including chat, text generation, summarization, and creating text embeddings.

The OCI Generative AI service includes the following foundational models:

- **Chat**

  Ask questions and get conversational responses through an AI chatbot.
- **Generation**

  Prompt the large language models (LLMs) to generate text or extract information from your text.
- **Summarization**

  Summarize text with your instructed format, length, and tone.
- **Embedding**

  Convert text to vector embeddings to use in applications for semantic searches, recommender systems, text classification, or text clustering.

  Integration of Generative AI Service with Siebel CRM currently supports the following types of use cases:

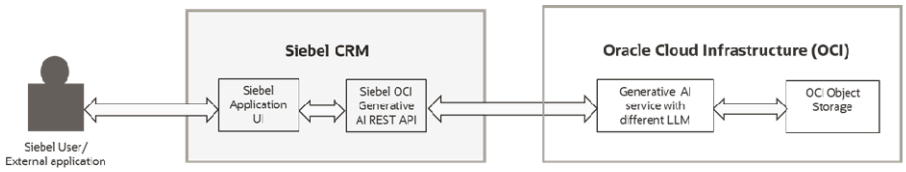
  - **Sentiment Analysis**

    Analyze the mood or tone of text and extract indicators of sentiment, such as positive, negative, or neutral. For more information, see *Use Case 1: Analyze Sentiment at Service Request and Activities Level*.
  - **Summarization**

    Generate executive summaries for documents that are too long to read, or summarize any type of text. For more information, see *Use Case 2: Generate Transfer and Closing Summary*.

Accessing Oracle Cloud Infrastructure (OCI) Generative Service

There are many ways to use and access OCI Generative Service - invoke use cases from Siebel UI/components like Workflow, Script/externally via REST API. You can use OCI Generative AI Service directly via Siebel REST API – for more information, see *Siebel REST APIs for Generative AI Service*.



As shown in this reference architecture, a typical flow for using OCI Generative AI Service is as follows:

1. On the Siebel CRM side:
    - You (user/external app) invoke the Siebel OCI AI Generative AI service via Siebel user interface (UI).
    - It can also be directly invoked via any Siebel component like Workflow or via any external self-service application layer (For example: Postman API client).
2. On the OCI side:
    - Siebel REST API does the required pre-processing, authenticates with OCI, and invokes the Generative AI Service via OCI Java SDK to get the required response using AI/ML.
    - Post-processing is done, and the required response is returned to the client.

Process of Setting Up Oracle Cloud Infrastructure (OCI) Generative AI Service

To set up and configure OCI Generative AI Service, perform the following tasks.

- *Enable Oracle Cloud Infrastructure (OCI) Generative AI Service*
- *Configure Authentication for Oracle Cloud Infrastructure (OCI) Generative AI Service*
- *Select Large Language Model*
- *Specify the Large Language Model (LLM) prompts*
- *Trigger use cases*
- *Configure and activate Runtime Event for Closing Summary*
- *Configure and activate Runtime Event for Transfer Summary*
- *Configure and activate Runtime Event for Sentiment Analysis at Activities Level*
- *Configure and activate Runtime Event for Sentiment Analysis at Service Request Level*
- *Configure Business Component User Properties for Summarization*
- *Configure Business Component User Properties for Sentiment Analysis*
- *Display Summary*
- *Configure Business Component User Properties for Multimodal AI*
- *Configure Business Component User Properties and activate Runtime Event for GSAT Score*

Enable Oracle Cloud Infrastructure (OCI) Generative AI Service

To enable Oracle Cloud Infrastructure (OCI) Generative AI Service

- Set the below System preference.

Name	Value
EnableOCIAIGenAI	N (to be changed to 'Y' to activate feature)

- By default, it is disabled.
- Generative AI use cases that are not required can be disabled .
- Create seed data as required for different use-cases (using the database setup wizard or through manual configuration).
- Select a model and configure parameters for prompts.
- OCI Generative AI provides a set of pre-trained models. You can create new custom models or create new versions of existing custom models with your own dataset.

Generative AI Service can be consumed through:

- On-demand (access pretrained LLMs and embedding models).
- Dedicated AI clusters (deploy pretrained LLMs or fine-tuned custom models exclusive to your tenancy).

Configure Authentication for Oracle Cloud Infrastructure (OCI) Generative AI Service

To configure authentication for OCI Generative AI Service

The following configurations for RunTime Event, BC User properties, Applet/View/Screen/Link/BO configurations are given for the OOTB use case. Please configure the same accordingly for your use cases.

1. Download the Authentication file from OCI which has the Credential details like user ID, fingerprint, and tenancy.
2. Rename the file `config` to `ociconfig.properties` and place it in the Siebel's internal Tomcat's webapps folder. The file should be a ‘properties’ file with the name as `ociconfig`.
3. Ensure that the `ociconfig.properties` file includes the location of the private key(.pem) file and make sure that the path mentioned is valid.
4. Import the necessary Certificates as required to both internal, external Tomcats and add it to the Siebel keystore or truststore.
5. Restart the Tomcats.

Select Large Language Model

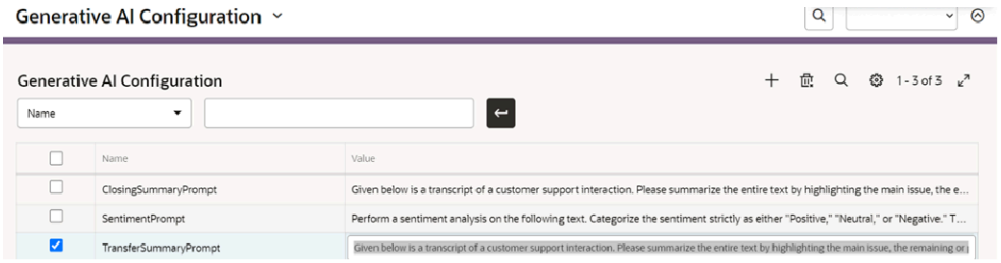
- Cohere Command R is the Out-of-the-box Large Language Model (LLM).
- Sentiment Analysis' and 'Summarization' use cases leverage the *Chat* functionality if COHERE command-r (cohere.command-r-XXXX) LLM is used.
- 'Summarization' use case leverages the *Summarize* functionality when COHERE command (cohere.command-r-XXXX) LLM is used.
- To change Generative AI Region and the corresponding endpoint (from US-CHICAGO-1), the below Business Service(*SiebelOCIAIWebService*) User Properties can be altered. Restart the Tomcat for the changes to be reflected.

BS User Property Name	Value
GenAIRegion	**<CHANGE_ME>** eg: AP-MUMBAI-1
GenAIRegionEndpoint	**<CHANGE_ME>** eg: <ocid.xxxxx>

Specify the Large Language Model (LLM) prompts

Following are specified LLM prompts:

- New View named *Generative AI Configuration List View* has been created with the below parameters to specify the LLM prompts:



- Configuration for command-r Large Language Model (LLM) is shown below:

Name	Value
ClosingSummaryPrompt	Given below is a transcript of a customer support interaction. Please summarize the entire text by highlighting the main issue, the exact device or account details, and the actions undertaken

Name	Value
	to resolve it. Ensure that the summary maintains the contextual history accurately and does not infer or create any conclusions about the customer's state of mind or satisfaction unless explicitly stated by the customer. Avoid adding any information that is not clearly present in the interaction.
TransferSummaryPrompt	Given below is a transcript of a customer support interaction. Please summarize the entire text by highlighting the main issue, the remaining or pending concern, the exact device or account details, and the actions undertaken to resolve it. Ensure that the summary maintains the contextual history accurately and does not infer or create any conclusions about the customer's state of mind or satisfaction unless explicitly stated by the customer. Avoid adding any information that is not clearly present in the interaction.
SentimentPrompt	Perform a sentiment analysis on the following text. Categorize the sentiment strictly as either Positive, Neutral, or Negative. The output of the sentiment analysis should definitely begin with one of these three words: Positive, Neutral, or Negative.

Note: Other custom parameters like ClosingSummaryPreambleOverride can be added in this section with corresponding entries in the BC User Property.

Trigger use cases

Following are some trigger use cases:

- Out-of-the box Generative AI use cases are controlled by Siebel Runtime Events.
  - Each use case (Sentiment Analysis / Summarization) can be individually Enabled/Disabled. By default, all the use cases are enabled.
  - Each of the two Summarization use-cases (Closing / Transfer Summary) can be Enabled/Disabled based on different Runtime events.
  - Summarization should have the Context set to *GenAI:SummarizeChatText* or *GenAI:Chat* while using command-R LLM.
  - Sentiment Analysis should have the Context set to *GenAI:GenerateChatText* or *GenAI:Chat* while using command-R LLM.
- Runtime Events work in conjunction with the Business Component User Properties in Out-of-the-box scenarios.
- Out-of-the-box Service Request and Action Business Component's have Runtime Events and corresponding Business Component User Properties.

Configure and activate Runtime Event for Closing Summary

Runtime Events with details of Event, Action Set and corresponding Action for Closing Summary is shown below:

Name	Active	Description
OCI Generative AI Service Closing Summary	True	Trigger when Activity Type is changed to Closing Summary

Actions to be populated as below which is used by Closing Summarization use cases while using the COHERE command-r (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
ClosingSummary	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:SummarizeChatText

Actions to be populated as below which is used by Closing Summarization use cases while using the COHERE command (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
ClosingSummary	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:SummarizeText

Events:

Seq	Object Type	Object Name	Event	Subevent	Conditional Expression	Action Set
1	BusComp	Action	SetFieldValue	Type	SystemPreference("EnableOCIAIGenAI") = "Y" AND [Type] = LookupValue ('TODO_TYPE' , 'CLOSING SUMMARY') AND GetProfileAttr('Me.ActiveViewName')='Service Request Detail View'	OCI Generative AI Service Closing Summary

Configure and activate Runtime Event for Transfer Summary

Runtime Events with details of Event, Action Set and corresponding Action for Transfer Summary is shown below:

Name	Active	Description
OCI Generative AI Service Transfer Summary	True	Trigger when Activity Type is changed to Transfer Summary

Actions to be populated as below which is used by Transfer Summarization use cases while using the COHERE command-r (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
TransferSummary	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:SummarizeChatText

Actions to be populated as below which is used by Transfer Summarization use cases while using the COHERE command (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
TransferSummary	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:SummarizeText

Events:

Seq	Object Type	Object Name	Event	Subevent	Conditional Expression	Action Set
2	BusComp	Action	SetFieldValue	Type	SystemPreference("EnableOCIAIGenAI") = "Y" AND [Type] = LookupValue ('TODO_ TYPE' , 'TRANSFER SUMMARY') AND GetProfileAttr('Me.ActiveViewName')='Service Request Detail View'	OCI Generative AI Service Transfer Summary

Configure and activate Runtime Event for Sentiment Analysis at Activities Level

Runtime Events with the details of the Event, Action Set and corresponding Action for Sentiment Analysis at Activities level is shown below:

Name	Active	Description
OCI Generative AI Service Activity Sentiment	True	Trigger when Activity Type is changed to Call - Inbound or Email - Inbound

Actions:

Actions to be populated as below which is used by Sentiment use cases while using the COHERE command-r (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
ActivitySentimen	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:GenerateChatText

Actions to be populated as below which is used by Sentiment use cases while using the COHERE command (`cohere.command-r-xxxx`) OR LLAMA 70b (`meta.llama-xxxx`).

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context
ActivitySentiment	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:GenerateText

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context

Events:

Seq	Object Type	Object Name	Event	Subevent	Conditional Expression	Action Set
3	BusComp	Action	SetFieldValue	Type	SystemPreference("EnableOCIAIGenAI") = "Y" AND ([Type] = LookupValue ('TODO_TYPE' , 'Call - Inbound') OR [Type] = LookupValue ('TODO_TYPE' , 'Email - Inbound')) AND GetProfileAttr('Me.ActiveViewName')='Service Request Detail View'	OCI Generative AI Service Activity Sentiment

Configure and activate Runtime Event for Sentiment Analysis at Service Request Level

Runtime Events with the details of the Event, Action Set and corresponding Action for Sentiment Analysis at Service Request level is shown below:

Name	Active	Description
OCI Generative AI Service SR Sentiment and Classification	True	Trigger when Service Request is saved

Actions to be populated as below which is used by Sentiment use cases while using the COHERE command-r (`cohere.command-r-xxxx`) LLM.

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context	Conditional Expression
ServiceRequestSentiment	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:GenerateChatText	SystemPreference("EnableOCIAIGenAI") = "Y"

Actions to be populated as below which is used by Sentiment use cases while using the COHERE command (`cohere.command-r-xxxx`) OR LLAMA 70b (`meta.llama-xxxx`).

Name	Action Type	Seq	Active	BS Name	BS Method	BS Context	Conditional Expression
ServiceRequestSentiment	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:GenerateText	SystemPreference("EnableOCIAIGenAI") = "Y"

Events :

Seq	Object Type	Object Name	Event	Subevent	Conditional Expression	Action Set
1	BusComp	Service Request	WriteRecord	N/A	[Description] IS NOT NULL	OCI Generative AI Service SR Sentiment and Classification

Configure Business Component User Properties for Summarization

The below configurations for RunTime Event, BC User properties, Applet/View/Screen/Link/BO configurations are given for the OOTB use case. Please configure the same accordingly for your use cases.

Business Component User Properties for Closing and Transfer Summary leveraging COHERE command-r LLM is shown here:

BC User Property Name	BC User Property Value
SummarizeChatText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
SummarizeChatText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
SummarizeChatText_Param_3	lovInfo:BC_FIELD=Type;LOV_TYPE=TODO_TYPE;LOV_LIC=CLOSING SUMMARY,TRANSFER SUMMARY [This parameter is used to pick the appropriate Localization value]
SummarizeChatText_Param_4	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
SummarizeChatText_Param_5	inputFromOtherBC:BUSCOMP=Service Request,FIELD=Abstract;BUSCOMP=Order Entry - Orders,FIELD=Order Priority,Account [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]
SummarizeChatText_Param_6	output:FIELD=Description;MAX_FIELD_LENGTH=250 [This is the BC field to which the LLM response will be populated. Please modify the MAX_FIELD_ LENGTH based on the respective BC Field length]
SummarizeChatText_Param_7	prompt:CLOSING SUMMARY=ClosingSummaryPrompt;TRANSFER SUMMARY=TransferSummaryPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
SummarizeChatText_Param_8	minimumInputCharLength:250 [This is the minimum input length to the LLM]

Business Component User Properties for Closing and Transfer Summary leveraging other Large Language Models (LLM) is shown here:

BC User Property Name	BC User Property Value
SummarizeText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
SummarizeText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
SummarizeText_Param_3	lovInfo:BC_FIELD=Type;LOV_TYPE=TODO_TYPE;LOV_LIC=CLOSING SUMMARY,TRANSFER SUMMARY [This parameter is used to pick the appropriate Localization value]
SummarizeText_Param_4	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
SummarizeText_Param_5	inputFromOtherBC:BUSCOMP=Service Request,FIELD=Abstract;BUSCOMP=Order Entry - Orders,FIELD=Order Priority,Account [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]
SummarizeText_Param_6	output:FIELD=Description;MAX_FIELD_LENGTH=250 [This is the BC field to which the LLM response will be populated. Please modify the MAX_FIELD_ LENGTH based on the respective BC Field length]
SummarizeText_Param_7	additionalCommand:CLOSING SUMMARY=ClosingSummaryPrompt;TRANSFER SUMMARY=TransferSummaryPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
SummarizeText_Param_8	minimumInputCharLength:250 [This is the minimum input length to the LLM]
SummarizeText_Param_9	extractiveness:CLOSING SUMMARY=ClosingSummaryExtractiveness;TRANSFER SUMMARY=TransferSummaryExtractiveness
SummarizeText_Param_10	format:CLOSING SUMMARY=ClosingSummaryFormat;TRANSFER SUMMARY=TransferSummaryFormat
SummarizeText_Param_11	length:CLOSING SUMMARY=ClosingSummaryLength;TRANSFER SUMMARY=TransferSummaryLength

Configure Business Component User Properties for Sentiment Analysis

The above configurations for RunTime Event, BC User properties, Applet/View/Screen/Link/BO configurations are given for the OOTB use case. Please configure the same accordingly for your use cases.

Business Component User Properties for Sentiment Analysis at Activities level leveraging COHERE command-r LLM is shown here:

BC User Property Name	BC User Property Value
GenerateChatText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
GenerateChatText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
GenerateChatText_Param_3	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
GenerateChatText_Param_4	prompt:SentimentPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
GenerateChatText_Param_5	output:FIELD=Sentiment [This is the BC field to which the LLM response will be populated.]
GenerateChatText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL [This parameter is used to pick the appropriate Localization value]
GenerateChatText_Param_7	outputToOtherBC:BUSCOMP=Service Request,FIELD=Sentiment [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]
PrivateFields	Modified Value: Service Address Name, Repeating Type LIC, **SentimentValue**

Business Component User Properties for Sentiment Analysis at Activities level leveraging other Large Language Models (LLM) is shown here:

BC User Property Name	BC User Property Value
GenerateText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]

BC User Property Name	BC User Property Value
GenerateText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
GenerateText_Param_3	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
GenerateText_Param_4	prompt:SentimentPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
GenerateText_Param_5	output:FIELD=Sentiment [This is the BC field to which the LLM response will be populated.]
GenerateText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL [This parameter is used to pick the appropriate Localization value]
GenerateText_Param_7	outputToOtherBC:BUSCOMP=Service Request,FIELD=Sentiment [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]
PrivateFields	Existing Value: Service Address Name,Repeating Type LIC Modified Value: Service Address Name,Repeating Type LIC,SentimentValue

Business Component User Properties for Sentiment Analysis at Service Request level leveraging COHERE command-r and other Large Language Models (LLM) is shown here:

BC User Property Name while using the COHERE command-r (`cohere.command-r-XXXX`) LLM	BC User Property Value
GenerateChatText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
GenerateChatText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
GenerateChatText_Param_3	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
GenerateChatText_Param_4	prompt:SentimentPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
GenerateChatText_Param_5	output:FIELD=Sentiment [This is the BC field to which the LLM response will be populated.]
GenerateChatText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL

BC User Property Name while using the COHERE command-r (`cohere.command-r-XXXX`) LLM	BC User Property Value
	[This parameter is used to pick the appropriate Localization value]
PrivateFields	SentimentValue

BC User Property Name while using the COHERE command (`cohere.command-r-XXXX`) OR LLAMA 70b (`meta.llama-XXXX`) LLM	BC User Property Value
GenerateText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
GenerateText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
GenerateText_Param_3	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
GenerateText_Param_4	prompt:SentimentPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
GenerateText_Param_5	output:FIELD=Sentiment [This is the BC field to which the LLM response will be populated.]
GenerateText_Param_6	lovInfo:LOV_TYPE=GENAI_SENTIMENT;LOV_LIC=POSITIVE,NEGATIVE,NEUTRAL [This parameter is used to pick the appropriate Localization value]
PrivateFields	SentimentValue

Display Summary

New Pop-Up Applet Generative AI Response Popup Applet with corresponding Business Component, Class and WebTemplate has been created to display the Summary.

Generative AI Response Popup Applet Configuration

Business Component

Name	Class	New / Existing
GenAI Response	CSSBCVGenAIPopup	New

Class

Name	DLL	New/Existing
CSSBCVGenAIPopup	SSCABCBC	New

New Business Service User Property has been created to refer to the Popup Applet.

Name	Value	New / Existing
GenAIPopUpApplet	Generative AI Response Popup Applet	New

Configure Business Component User Properties for Multimodal AI

To configure Business Component User Properties: Use Case 1

The above configurations for RunTime Event, BC User properties, Applet/View/Screen/Link/BO configurations are given for the OOTB use case. Please configure the same accordingly for your use cases.

1. **Required Seed Data changes:**
    a. Click on **Sitemap**.
    b. Navigate to **Administration- Application** > **Views**.
    c. Navigate to **Generative AI Configuration List** View and create a new record as per the following table:

Prompt Name	Prompt Value
ClosingSummaryPrompt	Given below is a transcript of a customer support interaction. The customer support interaction can contain an attached image or document as well. Please holistically summarize the entire text and the image data. The summary should highlight the main issue, the exact device or account details, and the actions undertaken to resolve it. Do mandatorily summarize the image or document if present. Ensure that the summary maintains the contextual history accurately and does not infer or create any conclusions about the customer's state of mind or satisfaction unless explicitly stated by the customer. Strictly avoid adding any information that is not clearly present in the interaction. Please look for the image or document attachment. The textual data is given below:
TransferSummaryPrompt	Given below is a transcript of a customer support interaction. The customer support interaction can contain an attached image or document as well. Please holistically summarize the entire text and the image data. The summary should highlight the main issue, the remaining or pending concern, the exact device or account details, and the actions undertaken to resolve it. Do mandatorily summarize the image or document if present. Ensure that the summary maintains the contextual history accurately and does not infer or create any conclusions about the customer's state of mind or satisfaction

Prompt Name	Prompt Value
	unless explicitly stated by the customer. Strictly avoid adding any information that is not clearly present in the interaction. Please look for the image or document attachment.

2. **Required Repository Changes:**

   a. Create Business Component User Properties.
   b. Add following Business Component User Properties while using the LLAMA (meta.llama-XXXX) LLM:

BC User Property Name	BC User Property Value
SummarizeChatText_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
SummarizeChatText_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
SummarizeChatText_Param_3	lovInfo:BC_FIELD=Type;LOV_TYPE=TODO_TYPE;LOV_LIC=CLOSING SUMMARY, TRANSFER SUMMARY [This parameter is used to pick the appropriate Localization value]
SummarizeChatText_Param_4	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
SummarizeChatText_Param_5	inputFromOtherBC:BUSCOMP=Service Request,FIELD=Abstract;BUSCOMP=Order Entry - Orders,FIELD=Order Priority,Account [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]
SummarizeChatText_Param_6	output:FIELD=Description;MAX_FIELD_LENGTH=250 [This is the BC field to which the LLM response will be populated. Please modify the MAX_FIELD_LENGTH based on the respective BC Field length]
SummarizeChatText_Param_7	prompt:CLOSING SUMMARY=ClosingSummaryPrompt;TRANSFER SUMMARY=TransferSummaryPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
SummarizeChatText_Param_8	minimumInputCharLength:250 [This is the minimum input length to the LLM]
SummarizeChatText_Param_9	runtimeType:LLAMA [This is the minimum input length to the LLM]

BC User Property Name	BC User Property Value
	Note: For Siebel CRM version 26.4 and above, replace “runtimeType” with “provider”.
SummarizeChatText_Param_10	imageUrl:BUSCOMP=Service Request Attachment,FIELD=ActivityFileName,SORT_SPEC=ActivityFileDate(DESC) [This is the input Field to specify the Image Attachment to the LLM]

Note: SummarizeChatText_Param (1-8) are available out of the box. Whereas, SummarizeChatText_Param (9-10) requires configuration.

3. **Modify `ociconfig.properties` file, make the following entry in the [GENAI] section:**

```
multimodal_image_type = png,jpg,jpeg
multimodal_max_image_size = 5242880
multimodal_image_for_summarize = true
```

To configure Business Component User Properties: Use Case 2

1. **Required Seed Data changes:**
   a. Click on **Sitemap**.
   b. Navigate to **Administration- Application** > **Views**.
   c. Navigate to **Generative AI Configuration List** View and create a new record as per the following table:

Prompt Name	Prompt Value
ExtractImagePrompt	Analyze the provided image and extract the address details. Extract the mentioned Address, City, State, Zip Code and the Country from the image. Do not extract any information which is not present in the image. The response should contain only the address, city, state, zip code and the country details. Display the attribute name, followed by a colon and its value and then leave a line space after each attribute. Do not include any other text in the response.

2. **Required Repository Changes:**
   a. Create Business Component User Properties for Service Request Attachment.

BC User Property Name	BC User Property Value
ExtractImageChat_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
ExtractImageChat_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]

BC User Property Name	BC User Property Value
ExtractImageChat_Param_3	prompt:ExtractImagePrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
ExtractImageChat_Param_4	imageUrl:BUSCOMP=Service Request Attachment,FIELD=ActivityFileName [This is the BC field whose input value will be send to the LLM]
ExtractImageChat_Param_5	runtimeType:LLAMA [Type or provider of the LLM like LLAMA] **Note:** For Siebel CRM version 26.4 and above, replace “runtimeType” with “provider”.
ExtractImageChat_Param_6	outputToOtherBC:BUSCOMP=GenAI Image Extract Response BC,FIELD=Address Maint - New Addr Line 1:Address,Address Maint - New City:City,Address Maint - New Country:Country,Address Maint - New State:State,Address Maint - New Zip Code:Zip Code; [This user property is used when we need to populate the output to other BC, apart from the current BC.]. FIELD=<BC Field>:<Param from LLM Response>
Named Method 1	"Extract", "INVOKESVC", "Service Request Attachment", "SiebelOCIAIService", "InvokeOCIAI", "'Context'", "'GenAI:ExtractImageChat'" [This is the BS Method invoked on click of the Button to perform functionality like Extracting content from an image]

3. **Modify `ociconfig.properties` file, make the following entry in the [GENAI] section:**

```
multimodal_image_type = png,jpg,jpeg
multimodal_max_image_size = 5242880
multimodal_image_for_summarize = true
```

Configure Business Component User Properties and activate Runtime Event for GSAT Score

Configure Seed Data for GSAT

The above configurations for RunTime Event, BC User properties, Applet/View/Screen/Link/BO configurations are given for the OOTB use case. Please configure the same accordingly for your use cases.

Follow these steps:

1. Click on **Sitemap**.
2. Navigate to **Administration- Application** > **Views**.
3. Navigate to **Generative AI Configuration List** View and create a new record as per the following table:

Prompt Name	Prompt Value
GSATPrompt	Provide the Agent Effort Score, Empathy Score, Customer Effort Score and Agent Knowledge Score Satisfaction Score by evaluating the following. Assign a score exactly in single digit from 1 to 5 for each criterion and should not be in fractions, with 1 being minimal effort and 5 being maximum effort. The scores should reflect the customer's feedback and comments. Also, provide Resolution Achievement(Yes or No) based on whether the customer's issue was fully resolved by the end of the interaction. Below is the customer's feedback:

4. Add responsibility to the **Service Request AI Info** View.

   Note: Make sure to launch application with `/editseeddata` for all Seed data changes.

   a. Click on **Sitemap**.
   b. Navigate to **Administration - Application** > **Views**.
   c. Create a new record as:

View	Description
Service Request AI Info View	Service Request AI Info View

   d. Add **Siebel Administrator** responsibility to this record using responsibilities child applet.
   e. Once responsibility is added, drilldown on **Siebel Administrator** which takes you to **Responsibilities** View.
   f. Click on **Clear cache** button.

Activate Runtime Event for GSAT

1. Click on **Sitemap**.
2. Navigate to **Administration- Runtime** Events.
3. Navigate to Action Sets view to find the below configuration:

Action Sets for GSAT use case

Name	Active	Description
OCI Generative AI Service GSAT Data	True	Trigger when SR status is changed to Closed

Actions to be populated as below which is used by GSAT use case

Name	Action Type	Sequence	Active	BS Name		
GSAT	BusService	1	True	SiebelOCIAIService	InvokeOCIAI	GenAI:G

Events:

Sequence	Object Type	Object Name	Event	Subevent	Conditional Expression	Action Set
1	BusComp	Service Request	SetFieldValue	Status	SystemPreference(" = "Y" AND [Status] = LookupValue ('SR_STATUS' , 'Closed') AND [Description] IS NOT NULL	OCI Generative AI Service GSAT Data

Creating a New Field

Create a new filed using the following values:

Name	Column	Calculated	Calc Value	Force Active	Length	Predefault Value	Postdefault Value	Type
Agent Effort Score	AGNT_EFRT_SCR	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_NUMBER
Agent Effort Score Reason	AGNT_EFRT_SCR_RN	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_TEXT
Agent Knowledge Score	AGNT_KNWLDG_SCR	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_NUMBER
Agent Knowledge Score Reason	AGNT_KNWLDG_SCR_RN	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_TEXT
Customer Effort Score	CSTMR_EFRT_SCR	N/A	N/A			N/A	N/A	DTYPE_NUMBER
Customer Effort Score Reason	CSTMR_EFRT_SCR_RN	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_TEXT
Empathy Score	EMPATHY_SCR	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_NUMBER
Empathy Score Reason	EMPATHY_SCR_RN	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_TEXT
Resolution Achievement	RSLTN_ACHVMNT	N/A	N/A	N/A	30	N/A	N/A	DTYPE_TEXT
Resolution Achievement Reason	RSLTN_ACHVMNT_RN	N/A	N/A	N/A	N/A	N/A	N/A	DTYPE_TEXT
Parent BC Name	N/A	True	ParentBCName	True	N/A	N/A	N/A	N/A
Parent Row Id	PAR_ROW_ID	N/A	N/A	N/A	15	Parent: 'Service Request.Id'	N/A	DTYPE_ID

Name	Column	Calculated	Calc Value	Force Active	Length	Predefault Value	Postdefault Value	Type
Parent Type Code	PAR_TYPE_CD	N/A	N/A	N/A	75	N/A	Field: 'Parent BC Name'	DTYPE_TEXT

To create new record under Business Object:

1. Query for the business object 'Service Request'.
2. Under Business Object Component, create new record as:

Business Component	Link
AI Information BC	Service Request/AI Information BC

Screen View

1. Query for the screen 'Service Request Screen'.
2. Under Screen View, create a new record as:

Name	View	Type	Parent Category	Sequence	View Text-String Reference	Menu Text-String Reference
Service Request AI Info View	Service Request AI Info View	Detail View	Service Request List	191	SBL_AI_ASSIST	SBL_AI_ASSIST

Configure Business Component User Properties for GSAT

Business Component User Properties for GSAT:

BC User Property Name	BC User Property Value
GenerateChatGSAT_Param_1	compartmentId: <CHANGE_ME> [Pick the value from OCI tenancy or OCI configuration file]
GenerateChatGSAT_Param_2	servingMode:ON_DEMAND [Possible values are ON_DEMAND or DEDICATED]
GenerateChatGSAT_Param_3	input:FIELD=Description [This is the BC field whose input value will be send to the LLM]
GenerateChatGSAT_Param_4	inputFromOtherBC:BUSCOMP=Action,FIELD=Description,Comment [This is the list of other BC fields whose input values will be send to the LLM along with that of the primary BC. Please note that all the other BC's should be part of the same BO]

BC User Property Name	BC User Property Value
GenerateChatGSAT_Param_5	prompt:GSATPrompt [This is the input prompt to the LLM which is/are defined in the Administration - Generative AI screen]
GenerateChatGSAT_Param_6	outputToOtherBC:BUSCOMP=AI Information BC,FIELD=Agent Effort Score:Agent Effort Score,Agent Knowledge Score:Agent Knowledge Score,Customer Effort Score:Customer Effort Score,Empathy Score:Empathy Score,Resolution Achievement:Resolution Achievement [This is the BC field to which the LLM response will be populated. Please modify the MAX_FIELD_LENGTH based on the respective BC Field length]
Always Enable Child: AI Information BC	TRUE

Generative AI Service Use Cases

This topic describes the following sample Generative AI use cases:

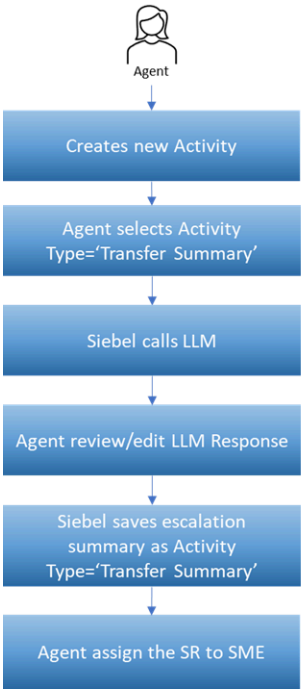
- *Use Case 1: Analyze Sentiment at Service Request and Activities Level*.
- *Use Case 2: Generate Transfer and Closing Summary*
- *Use Case 3: Multimodal AI*
- *Use Case 4: Generated Satisfaction Score*

Use Case 1: Analyze Sentiment at Service Request and Activities Level

In this scenario, Generative AI Service extracts the individual aspects in the inbound communication (e-mail, call) received by a call center agent and classifies each of the aspects into one of the polarity classes - positive, negative, and neutral. You can perform Sentiment Analysis at Service Request level and Activities level. However, Sentiment at Activities level takes precedence over Service Request level during analysis.

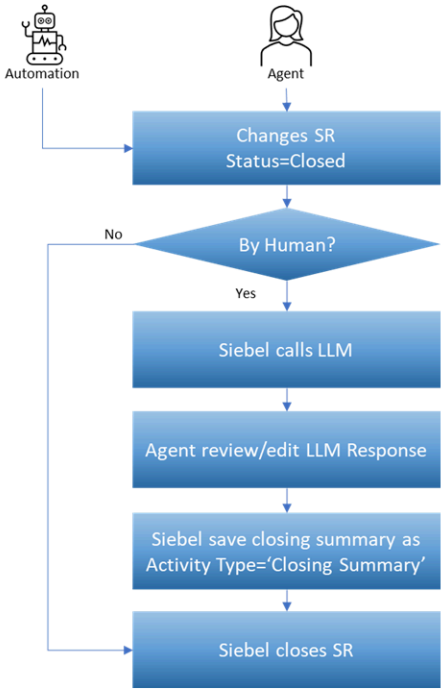
Use Case 2: Generate Transfer and Closing Summary

In this scenario (illustrated in the following image), Generative AI automatically generates a Summary of the conversation, highlighting key points, action items, and resolutions. Transfer Summary for another Agent is generated in case Call Center Agent needs to escalate/transfer Service Request to another Agent and a Closing Summary to be sent to customer is generated in case of a successful resolution.



As shown in the above image, a typical flow involved in generating a Transfer Summary is a follows:

1. User needs to create new activity and select `Type=Transfer` Summary.
2. Siebel will trigger call to the configured Large Language Model to generate summary based on all inbound and outbound communications.
3. User will have an opportunity to review and edit response before saving Transfer Summary and assigning Service Request to another agent.



As shown in the above image, a typical flow involved in generating a Closing Summary is a follows:

1. User needs to create new activity and select `Type=Closing` Summary.

2. Siebel will trigger call to the configured Large Language Model to generate summary based on all inbound and outbound communications between call center agent and customer.
3. User will have an opportunity to review and edit response before saving Closing Summary.

Use Case 3: Multimodal AI

In this scenario, Generative AI integrates and processes multiple types of data such as text, and images—to generate more comprehensive insights and responses. This bridges the gap between vision and language, interpreting the context, and delivering results in a contextually relevant response.

Use Case Scenario 1: Closing or Transfer Summary

After completing all the work associated with a Service Request and issue is resolved, Agent can generate a Closing Summary. Transfer Summary for another Agent is generated in case Agent needs to transfer Service Request to another Agent.

- Create new activity and select `Type=Closing Summary / Type=Transfer Summary`.
- Drawer window displays the `Closing / Transfer Summary`.
- User will have an opportunity to review and edit response before saving `Closing / Transfer Summary`.

Note:

1. After a new Closing / Transfer Summary activity is selected, Siebel will trigger call to the configured Large Language Model (LLM) to generate a Closing / Transfer Summary based on all inbound, outbound communications and Service Request Attachments (images)
2. Currently, only the last uploaded Attachment(image) in the Service Request Attachment Applet is sent to the Large Language Model (LLM)
3. The file formats supported are JPEG, JPG, PNG

Use Case Scenario 2: Creating a Service Request

While entering the necessary fields associated with a Service Request, Agent can use AI model to to automatically detect predefined values, extract text from documents uploaded and auto-populate in the respective field

- In the Service Requests list, create a new record and save
- In the Attachments applet, click Add File / Add URL
- Click Extract button
- Pre defined values are auto-populated in the respective fields
- User will have an opportunity to review and edit before saving

Note:

1. After Extract button is clicked, Siebel will trigger call to the configured Large Language Model (LLM) to extract text, and other key data from the image
2. Currently, you can pass only a single image per Large Language Model (LLM) call for extraction
3. The file formats supported are JPEG, JPG, PNG

Configuration and Authentication

To configure Generative AI capabalities

- To enable and configure Generative AI capabilities, set the following System Preference:

Name	Vaue
EnableOCIAIGenAI	N (to be changed to 'Y' to activate feature)

> **Note:** By default it is disabled.

- Create seed data, as required (using the database setup wizard or through manual configuration)

To authenticate Generative AI

- Download the Authentication file from OCI which has the Credential details like userId, fingerprint, and tenancy.
- Rename the file `config` to `ociconfig.properties` and place it in Siebel's internal Tomcat's webapps folder.
- Ensure that the `ociconfig.properties` file includes the location of the private key (`.pem`) file.
- Import the necessary certificates as required to both internal and external Tomcats and add it to the Siebel Keystore and Truststore.

Use Case 4: Generated Satisfaction Score

In this scenerio, Generative AI produces intelligent real-time insight by aggregating across a variety of important customer satisfaction indicators. Indicators like, Customer Effort, Agent Effort, Agent Knowledge, and Empathy are evaluated after every interaction or purchase. It is measured on a 5 point scale, where 1 indicates minimal effort and 5 represents maximum effort.

Configuration and Authentication

To configure Generative AI capabalities

- To enable and configure Generative AI capabilities, set the following System Preference:

Name	Vaue
EnableOCIAIGenAI	N (to be changed to 'Y' to activate feature)

> **Note:** By default it is disabled.

To authenticate Generative AI

- Download the Authentication file from OCI which has the Credential details like userId, fingerprint, and tenancy.

- Rename the file `config` to `ociconfig.properties` and place it in Siebel's internal Tomcat's webapps folder.
- Ensure that the `ociconfig.properties` file includes the location of the private key (`.pem`) file.
- Import the necessary certificates as required to both internal and external Tomcats and add it to the Siebel Keystore and Truststore.

Siebel REST APIs for Generative AI Service

Generative AI REST API can be invoked from any external client or internal Siebel artifacts like Workflow, Business Service, and script.

- SiebelOCIAIWebService supports the Generative AI capabilities (existing Service has been enhanced for Generative AI).
- Business Service Access to be given for *SiebelOCIAIWebService* with correct Responsibility.
- REST API endpoint with verb as POST is as below:

  `https://<hostname>:<port>/siebel/v1.0/service/SiebelOCIAIWebService/InvokeOCIAI`

- Authentication credential is same as that of the Siebel environment (mentioned in the 'hostname' of the REST endpoint).
- Supported functionalities with different Large Language Models are Chat (GenerateChatText / SummarizeChatText), GenerateText, SummarizeText, and EmbedText.

Following topics are a part of this topic:

- *REST API for Chat*
- *REST API for Generate Text*
- *REST API for Summarize Text*
- *REST API for Embed Text*

REST API for Chat

```
//Request payload with COHERE command-r
{
 "body": {
 "Context": "GenAI:Chat", //OR "GenAI:GenerateChatText" OR "GenAI:SummarizeChatText"
 "Chat_Param_1": "prompt:Write some facts about Oracle India Pvt Ltd" ,
 "Chat_Param_2": "compartmentId:ocid1.tenancy.oc1..XXXXXX",
 "Chat_Param_3" : "runtimeType:COHERE",
 Note: For Siebel CRM version 26.4 and above, replace "runtimeType" with "provider".
 "Chat_Param_4" : "maxTokens:3500",
 "Chat_Param_5" : "temperature:0",
 "Chat_Param_6" : "servingMode:ON_DEMAND",
 "Chat_Param_7" : "isEcho:true",
 "Chat_Param_8" : "isStream:false"
 }
}

//corresponding Response Payload
{
 "Status": "Success",
 "Output": "CohereChatResponse(super=BaseChatResponse(super=BmcModel(__explicitlySet__=[finishReason,
 chatHistory, text, prompt])), text=1. Oracle India Pvt Ltd is the Indian subsidiary of the American
```

```
 multinational computer technology company Oracle Corporation.\n\n2. Oracle India has its headquarters in
 Bangalore, Karnataka. The facility spans across 9 acres and is one of the largest owned by an international
 software company in India.\n\n3. The Indian subsidiary was established in 1994 and has played a significant
 role in Oracle's global strategy.\n\n4. Oracle India Pvt Ltd is a significant research and development hub
 for the company, focusing on product development, support, and management. It has contributed significantly
 to various Oracle products and technologies.\n\n5. The company has multiple offices across India, including
 locations in Gurgaon, Hyderabad, Mumbai, and New Delhi, employing thousands of professionals.\n\n6. Oracle
 India offers a wide range of job opportunities, from software development and engineering to sales,
 marketing, and customer support.\n\n7. It has a strong focus on training and development, providing
 extensive opportunities for employees to enhance their skills and grow professionally.\n\n8. Oracle India
 actively engages in corporate social responsibility initiatives. It has partnered with NGOs and local
 communities to support education, environment sustainability, and social empowerment programs. . . . .",
 "OPCRequestId": "21F32E8D687AAAAAAA/BE22BA603BB123C5E5BBBBBBB/FCCDA06F70EF3CCFCCCCCCCC"
}
```

REST API for Generate Text

```
//Request payload with cohere.command-r-XXXX or LLAMA 70b
{
 "body": {
 "Context": "GenAI:GenerateText",
 "GenerateText_Param_1": "prompt:Write an essay about Indian Sub continental summer in the style of
 Rabindranath Tagore" ,
 "GenerateText_Param_2": "compartmentId:ocid1.tenancy.oc1..XXXXXXX",
 "GenerateText_Param_3" : "runtimeType:COHERE",
 Note: For Siebel CRM version 26.4 and above, replace "runtimeType" with "provider".
 "GenerateText_Param_4" : "maxTokens:2000",
 "GenerateText_Param_5" : "temperature:1"
 }
}

//corresponding Response Payload
{
 "Status": "Success",
 "Output":
 "CohereLlmInferenceResponse(super=LlmInferenceResponse(super=BmcModel(__explicitlySet__=[generatedTexts,
 timeCreated])), generatedTexts=[GeneratedText(super=BmcModel(__explicitlySet__=[id,
 text])id=6f9e7893-4456-4766-89aa-c08d43b04476, text= The Indian Subcontinental summer arrives with a
 vibrant explosion of life, a dazzling feast for the senses that leaves one utterly enchanted. The air is
 thick with the fragrant scent of blossoming flowers, their vivid colors painting the landscape in vibrant
 hues. As the sun rises, it casts a golden glow upon the earth, awakening every creature and nurturing
 the fertile soil that lies beneath.\n\nIn this sultry season, the days stretch on forever, enveloping
 all in an embrace of relentless warmth. The relentless beating of the sun's rays on the earth's surface
 conjures forth an incessant chorus of shimmering heat waves that rise and fall like a rhythmic symphony.
 Yet, amidst this engulfing warmth, nature reveals her ingenious ways of offering solace.\n\nTrees, lush and
 leafy, provide a much-needed refuge from the piercing sun. Their outstretched branches sway gently in the
 breeze, inviting passersby to seek shelter beneath their leafy canopy. Here, in this natural sanctuary, a
 cool breeze whispers through the foliage, carrying with it a refreshing aroma that invigorates the soul.
\n . . . . . , likelihood=null, finishReason=null, tokenLikelihoods=null)], timeCreated=Wed Jun 26 03:56:25
 PDT 2024, prompt=null)",
 "OPCRequestId": "21F32E8D687AAAAAAA/BE22BA603BB123C5E5BBBBBBB/FCCDA06F70EF3CCFCCCCCCCC"
}
```

REST API for Summarize Text

```
//Request payload with cohere.command-r-XXXX
{
```

```
 "body": {
 "Context": "GenAI:SummarizeText",
 "SummarizeText_Param_1": "input:Customer says Hi, I recently upgraded my plan to include international
 calling, but I'm unable to dial overseas numbers. I've checked the documentation and made sure that the
 number I'm dialing is correct and follows the proper format, but it still isn't working. Can you please
 help me troubleshoot? Support says Yes, of course. International calling can sometimes be tricky. I would
 love to help you with this matter. To get started, may I please have your account number and phone number
 associated with the plan. Customer says Account is 123456 and phoneis 987-654-3210. Support says Thank
 you. I see that you have an outstanding balance on your account which could be causing the restriction on
 international calling. To resolve this, you need to pay the balance upfront. Unfortunately, I am unable
 to process payments, but I'll transfer you to our payment support team who will be able to assist you
 further and remove the restriction once the payment is complete.Customer says Can you please transfer
 me immediately? I really need to make this call today.Support says Absolutely! Our payment support team
 will be better equipped to help you with your urgent matter. Please remain on the line, and you'll be
 connected to an available agent shortly. If you get disconnected accidentally, you can reach them directly
 at 456-7890 ext. 2345. We apologize for any inconvenience this has caused." ,
 "SummarizeText_Param_2": "compartmentId:ocid1.tenancy.oc1..XXXXXX"

 }


//corresponding Response Payload
{
 "Status": "Success",
 "Output": "The customer recently upgraded their plan to include international calling, but they are unable
 to dial overseas numbers. The support representative asks for the customer's account number and phone
 number (123456 and 987-654-3210 respectively) and determines that the customer has an outstanding balance
 on their account, which is causing the restriction. To resolve this, the representative can't process the
 payment themselves, so they transfer the customer to the payment support team who can assist them in paying
 the balance and removing the restriction. The customer is transferred immediately and given a direct line
 to the payment support team in case they get disconnected. \n\nWould you like me to help the customer with
 any other issues?",
 "OPCRequestId": "21F32E8D687AAAAAAA/BE22BA603BB123C5E5BBBBBBB/FCCDA06F70EF3CCFCCCCCCCC"
}
```

REST API for Embed Text

```
//Request payload with cohere.command-r-XXXX
{
 "body": {
 "Context": "GenAI:EmbedText",
 "EmbedText_Param_1": "prompt:If the problem still persists, might need to debug the code. Please set
 Immediate Post to True for the field Interest in the event. If it did not resolve the issue,try to activate
 the Total field in the quote BC. This is the new SR in your queue." ,
 "EmbedText_Param_2": "compartmentId:ocid1.tenancy.oc1..XXXXXXX"

 }
}


//corresponding Response Payload
{
 "Status": "Success",
 "Output": "EmbedTextResult(super=BmcModel(__explicitlySet__=[embeddings, modelId, modelVersion,
 id])id=f16ba6a6-0368-47a4-9384-0089a83064f2, inputs=null, embeddings=[[1.3017578, 0.011962891, -0.39868164,
 -1.3125, 1.1777344, -1.2578125, -1.9111328, -1.7558594, 2.2617188, -1.8447266, 2.2578125, -0.76708984,
 -2.15625, -2.1992188, 1.6982422, 0.82177734, -1.0625, -0.32788086, 0.17468262, -0.2487793, 0.15112305,
 0.59716797, 3.0429688, 0.11425781, -0.54589844, 1.3798828, 1.2939453, 0.7680664, -1.0263672, 0.9848633,
 0.98046875, -0.36328125, -0.6352539, 0.33325195, -1.6826172, 0.03955078, -3.0664062, 0.24401855,
 -2.1816406, 2.6816406, -1.2578125, 0.77197266, -1.0341797, 0.088378906, -0.5395508,
 . . .
```

```
 25, 1.1748047, 0.53027344, 0.019500732, 0.79589844, 1.2939453, 1.6904297, -0.50146484, -0.1138916,
 -0.9296875, 1.2880859, -0.85302734, 0.69677734, -2.9550781, 1.9208984, 0.82666016]], modelId=cohere.embed-
english-light-v2.0, modelVersion=2.0)",
 "OPCRequestId": "21F32E8D687AAAAAAA/BE22BA603BB123C5E5BBBBBBB/FCCDA06F70EF3CCFCCCCCCCC"
}
```

4 Agentic AI

Retrieval Augmented Generation (RAG)

This topic describes some sample Retrieval Augmented Generation (RAG) use cases and how to setup and configure RAG for Siebel CRM.

About Retrieval Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) improves the quality of responses generated by a Large Language Model (LLM) by enriching prompts with relevant, up-to-date, and domain-specific information retrieved from enterprise data sources before the request is sent to the model. This approach enables more accurate and context-aware responses without requiring fine-tuning of the underlying LLM.

Within Siebel CRM, RAG can leverage contextual information from Siebel database records, Siebel File System attachments, Knowledge Management (KM) repositories, Oracle Cloud Infrastructure (OCI) Object Storage, and other enterprise content repositories. By grounding AI responses in an organization's own data, RAG helps users access more relevant and trustworthy information directly from their Siebel environment.

Key benefits of RAG include:

- Context-rich responses that incorporate customer-specific CRM data rather than relying solely on the LLM's general knowledge.
- Support for both structured and unstructured data, including Siebel business records, documents, attachments, and KM articles.
- Advanced similarity search capabilities across Service Requests and related business objects through a combination of semantic search, keyword matching, and structured filtering.
- Improved answer relevance for service, sales, and administrative scenarios without the cost and complexity of model fine-tuning.
- Deployment flexibility, with support for OCI-based retrieval services in cloud environments and OpenSearch-based retrieval for on-premises implementations.

Using Retrieval Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) in Siebel CRM operates in two phases, design time and run time. During the design-time phase, data is processed, prepared, and indexed for efficient retrieval. During the run-time phase, the

user's query is enhanced with relevant context retrieved from the indexed data, enabling the system to generate a more accurate and contextual response.

Design Time for RAG

To prepare data for RAG at design time:

1. Identify the source data required by the AI system. This may include unstructured content such as documents, files, and attachments, as well as structured data stored in systems like Siebel databases.
2. Prepare unstructured content by cleaning, segmenting, and chunking it into manageable pieces. Applying overlapping chunks can improve retrieval accuracy, particularly for lengthy documents.
3. Generate vector embeddings for the processed content using the same embedding model that will be employed during runtime query processing.
4. Store the generated embeddings in the designated vector repository. For cloud deployments, use Oracle Database 23ai Vector Search or OCI Knowledge Bases; for on-premises deployments, use Siebel OpenSearch.
5. For structured Siebel data, retain key attributes such as Id, Status, Priority, Date, Account, and Revenue as metadata or filtering criteria. Only descriptive fields, such as Summary and Description, should be converted into text for embedding generation.
6. Optionally, extract entities and relationships from both structured and unstructured data sources to enable graph-based retrieval and enhance relationship-aware search capabilities.

Run Time for RAG

To process a query using RAG at runtime:

1. Initiate a query or prompt from a user, business service, workflow, script, API, or assistant.
2. Convert the query into an embedding using the same embedding model employed during the design phase.
3. Perform a similarity search against the vector store to identify the most relevant content. In on-premises deployments, OpenSearch uses hybrid search with configurable weighting. In cloud deployments, RAG uses the OCI RAG Agent chat workflow.
4. Combine the retrieved content with the original query to enrich the prompt with relevant context.
5. Submit the enhanced prompt to the selected large language model (LLM), such as an OCI Generative AI model or another configured LLM.

   The LLM generates a response, which is returned to the originating channel, including the Siebel UI, workflow, business service, eScript, REST API, or any custom application that initiated the request.

Prerequisites

Make sure you the following prerequisites are met:

- OCI AI Services and Generative AI Services configuration must be completed and validated.
- The `ociconfig.properties` file must be present in the internal Tomcat webapps directory and configured with the required OCI parameters, including tenancy, user, fingerprint, region, compartment, and private key file information.
- When the KnowledgeBaseType parameter is set to OpenSearch, OpenSearch must be installed, configured, and operational.
- All data categories and business components intended for search must be enabled for indexing, and the initial Index or Index All process must have completed successfully.
- Required Siebel seed data, business services, applets, views, manifest files, and repository objects must be available in the environment or migrated prior to deployment.

- End users and administrators must have the necessary access privileges to applicable views, buttons, business services, and any newly introduced AI or RAG-related business services.

Process of Setting Up Retrieval Augmented Generation (RAG)

To set up and configure Retrieval Augmented Generation, perform the following tasks.

1. Enable the RAG system preferences.
2. Configure the knowledge base type for the deployment.
3. Configure OCI authentication, Generative AI, or OpenSearch as required.
4. Configure prompts and prompt augmentation templates.
5. Configure or verify the RAG-specific Siebel repository and manifest records.
6. Index or ingest the required data sources.
7. Validate the runtime flow from the Siebel UI or through a business service/API call.

Retrieval Augmented Generation (RAG) AI Use Cases

This section describes the following sample Retrieval Augmentation Generation (RAG) use case:

- *Use Case 1: Service Request (SR) Similarity Search*

Use Case 1: Service Request (SR) Similarity Search

This topic explains how Service Agents can identify Service Requests in the queue that are similar to existing ones.

Service Request Similarity Search helps service agents identify existing Service Requests (SRs) that are similar to the current SR. The search can use the SR description, summary, type, area, sub-area, attachment summaries, activity comments, and configured custom fields.

Typical User Workflow

1. Open the Service Request record that requires investigation.
2. Click **View Solutions** in the SR Information applet.
3. Review the Similar SRs and related Knowledge Articles, if KM integration is enabled.
4. Use the result hyperlinks and summaries to determine whether an existing SR is relevant.
5. Click **View Similar SRs** to open the results page when a more detailed list of results is required.
6. Open an SR hyperlink to review the full Service Request details.
7. Click **Link to Service Request** to associate a relevant existing SR with the current SR.

OpenSearch-Based Process

1. Summarize the current SR, including configured attachment and related BC context.
2. Send the summarized text and relevant metadata to OpenSearch.
3. Use hybrid search to retrieve similar SRs from indexed Siebel data.
4. Augment the retrieved results with additional context.
5. Display the enriched results in the Siebel UI.

OCI-Based Process

1. Summarize the current SR, including attachments and configured BC field data.
2. Send the summarized context to the configured OCI knowledge base or RAG Agent flow.

3. Retrieve related documents or records and augment the prompt.
4. Use the LLM to generate a contextual response.
5. Display the generated response and links to the user.

Troubleshooting and Best Practices

Symptom	Checks
RAG UI or buttons are not visible	Validate RAG configuration, manifest expressions, renderer file mappings, view ownership and responsibilities, and browser cache behavior.
Similar SR results are missing or incomplete	Confirm that OpenSearch is enabled, the Service Requests – Modern Search category is indexed, and all required SR fields are present in the indexed content.
Cloud RAG response is not returned	Validate OCI connectivity and configuration, including authentication settings, compartment ID, region, key file, bucket/knowledge base setup, agent endpoint, and network access.
Attachment context is not included	Verify Service Request Attachment settings, file accessibility, `SummarizeAttachmentChat` user properties, and support for the required document and image formats.
Responses are generic or not grounded	Review prompt templates, chunking strategy, embedding model alignment, result limits, hybrid search weighting, and data freshness to ensure optimal retrieval quality.

Best Practices

- **Security and Governance**
  - Index only data that users are authorized to access. Since RAG does not enforce Siebel data access controls, customer security requirements, user responsibilities, and data retention policies must be taken into account when selecting data sources.
- **Knowledge Base Configuration**
  - Use the same embedding model for both design-time ingestion and runtime query embedding within a given knowledge base to ensure embedding consistency.
  - Store structured data as metadata or filters, while embedding descriptive text fields to support effective semantic search.
  - Evaluate and optimize chunk size, overlap, and contextual chunking configurations for large documents to enhance retrieval accuracy and relevance.
- **Search Optimization**
  - Fine-tune hybrid search parameters, including search weights, result limits, and prompt augmentation templates, in a non-production environment before deployment.
- **Monitoring and Maintenance**
  - Continuously monitor RAG-generated responses for accuracy, completeness, and proper grounding against source content. Refine data preparation processes, prompts, and retrieval configurations as needed.
  - Re-index or refresh the knowledge base whenever significant source data changes occur or when additional data categories are introduced.

Siebel AI Connectors

This topic provides an overview of Siebel AI Connectors. Siebel AI Connectors enable customers to expose selected Siebel Open Integration REST APIs as Model Context Protocol (MCP)-compatible tools and servers for consumption by AI agents and assistant applications.

Siebel Open Integration exposes Siebel objects, services and workflows through REST APIs. These APIs are powerful and flexible for direct system-to-system integration. However, AI agents require a more structured interface for interacting with enterprise systems.

Siebel AI Connectors provide a configuration-driven framework for exposing selected Open Integration REST APIs as MCP tools. Customers identify the Open Integration REST APIs they want to expose, define MCP tool metadata & maintain a centralized catalog, group those tools into MCP Servers, and once configured deploy these servers either on-premises or via Siebel Cloud Manager(SCM). The Siebel AI Connectors runtime then realizes the configured MCP Servers and exposes runtime endpoints that AI agents can use to discover and invoke the configured tools.

By using Siebel AI Connectors, customers can expose only the Open Integration operations that are relevant for an AI-agent use case, keep access to Siebel data and actions controlled, and align agent behavior with business processes.

This topic covers the following sections:

- *Installing Siebel AI Connectors*
- *Configuring Siebel AI Connectors*
- *Deploying Siebel AI Connectors*

Installing Siebel AI Connectors

This topic describes how to install Siebel AI Connectors.

Siebel AI Connectors can work with on-premises environments and in Kubernetes-based environments managed through Siebel Cloud Manager. The deployment model determines how the runtime package and configuration artifacts are delivered, but the core runtime behavior remains the same.

In an on-premises deployment, the runtime package includes the artifacts required to run Siebel AI Connectors in the target environment.

In a Siebel Cloud Manager deployment, the Siebel AI Connectors runtime is deployed in a Kubernetes-based environment. Configuration artifacts are provided as part of the deployment process. More details to how we can work with Siebel Cloud Manager to deploy and manage Siebel AI Connectors can be found in the Deploying Siebel AI Connectors section of this topic. For more details on SCM, please read the Administration Bookshelf guide for Siebel Cloud Manager.

> **Note:**
> - Open Integration installation and configuration is a prerequisite to ensure that your Siebel AI Connectors works properly.
> - During the installation process, don't modify the runtime JAR or dependent libraries. Update only the configuration files.

On-Premises Package Contents

The on-premises runtime package includes the runtime JAR, dependent libraries, and configuration template files.

The package includes the following artifacts:

```
mcp-runtime/
  ├── mcp-runtime-<version>.jar. # Do not Modify
  ├── libs/ # Do not Modify
  ├── application.yaml # Configuration File
  ├── logging.properties # Configuration File
  ├── MCP_CONFIG.json # Configuration Files
  ├── FINAL_SPECIFICATION.json # Configuration Files
  ├── OI_CONFIG.json # Configuration Files
```

On-Premises Installation for Windows & Linux

Windows Installer

Interactive Installation

Use this procedure to install Siebel AI Connectors on Windows by using the interactive installer.

1. Download and extract the Siebel AI Connectors installer artifacts.
2. Review the README file provided with the installer package.
3. Start the installer by running the Windows installer command.
4. Select the installation directory.
5. Select the Siebel AI Connectors component.
6. Review the installation summary.
7. Click **Install**.
8. After installation completes, verify that the Siebel AI Connectors artifacts are created in the selected installation directory.
9. Configure the runtime artifacts before starting the Siebel MCP Server.

Silent Installation

Use this procedure to install Siebel AI Connectors on Windows in silent mode.

1. Run the installer and proceed to the Summary screen.
2. Save the response file.
3. Review the response file and verify that the installation directory is correct.
4. Run the installer in silent mode using the generated response file.
5. After installation completes, verify that the Siebel AI Connectors artifacts are created in the selected installation directory.

Linux Installer

Interactive Installation

Use this procedure to install Siebel AI Connectors on Linux by using the interactive installer.

1. Download and extract the Siebel AI Connectors installer artifacts.
2. Review the README file provided with the installer package.
3. Start the installer by running the Linux installer command.
4. Select the installation directory.
5. Select the Siebel AI Connectors component.
6. Review the installation summary.

7. Click **Install**.
8. After installation completes, verify that the Siebel AI Connectors artifacts are created in the selected installation directory.
9. Configure the runtime artifacts before starting the Siebel MCP Server.

Silent Installation

Use this procedure to install Siebel AI Connectors on Linux in silent mode.

1. Run the installer and proceed to the Summary screen.
2. Save the response file.
3. Review the response file and verify that the installation directory is correct.
4. Run the installer in silent mode using the generated response file.
5. After installation completes, verify that the Siebel AI Connectors artifacts are created in the selected installation directory.

Configuring Siebel AI Connectors

This section describes how to configure Siebel AI Connectors, prepare MCP tools from Siebel Open Integration REST APIs, secure MCP endpoints, and validate runtime behavior.

Siebel AI Connectors are intended to be used with Siebel Open Integration. After Open Integration is configured and running, it exposes the resources defined for API generation under `config.json` as standard REST APIs. The corresponding OpenAPI specification available from the Open Integration runtime describes the exposed resources, paths, HTTP methods, parameters, request bodies, and schemas.

Customers use the Open Integration REST APIs and the corresponding OpenAPI specification as the starting point to decide which operations should be selectively exposed as MCP tools. The selected operations and tool metadata are represented in `FINAL_SPECIFICATION.json`, grouped into MCP Servers using `MCP_CONFIG.json`, connected to Open Integration using `OI_CONFIG.json`, and loaded by the runtime using `application.yaml` and `logging.properties`.

Configuration Prerequisites

Before configuring or deploying Siebel AI Connectors, complete the following prerequisites.

Open Integration

Siebel Open Integration must be installed, configured, and running before Siebel AI Connectors can be configured.

Check	Description
Open Integration is installed and configured.	Siebel AI Connectors depend on Open Integration REST APIs as the source APIs for MCP tools.
Open Integration runtime is running.	The runtime must expose the configured resources as standard REST APIs.
OpenAPI specification is available from the Open Integration runtime.	The OpenAPI specification describes the resources, paths, HTTP methods, parameters, request bodies, and schemas available for tool creation.
Target Open Integration server details are known.	These values are required in `OI_CONFIG.json`, including server name, scheme, host, port, and context.

Tool Selection

Before creating or updating `FINAL_SPECIFICATION.json`, identify which Open Integration operations should be exposed as MCP tools.

Check	Description
Required business use case is identified.	Only expose APIs required by the intended AI-agent use case.
Required Open Integration resources are identified.	For example, Account, Contact, Product, Service Request, or other configured resources.
Required API operations are selected.	Select the required paths and HTTP methods, such as **`get`**, **`post`**, **`put`**, or **`delete`**.
Tool names and descriptions are planned.	Tool names should be stable, and descriptions should be clear enough for MCP clients and AI agents.
Required request inputs are understood.	Review path, query, header parameters, and request body schemas.
Security impact is reviewed.	Avoid exposing create, update, or delete operations unless the AI-agent use case requires them.

Runtime Package

The Siebel AI Connectors runtime package must be available on the target environment.

Check	Description
`mcp-runtime-<version>.jar` is present.	Runtime application binary. Do not modify.
`libs/` is present.	Runtime dependencies. Do not modify.
`application.yaml` is present.	Defines runtime server settings, security settings, and configuration file locations.
`logging.properties` is present.	Defines runtime logging behavior.
`MCP_CONFIG.json` is present.	Defines MCP Servers and tool mappings.
`FINAL_SPECIFICATION.json` is present.	Defines selected tools in OpenAPI-compatible format.
`OI_CONFIG.json` is present.	Defines Open Integration server instances.

Security

If JWT authentication is enabled for Siebel AI Connectors, verify the following:

Check	Description
Identity provider is configured.	For example, OCI Identity Domain or the configured OAuth provider.
JWT issuer is known.	Required in **`security.jwt.issuer`**.
JWT audience is known.	Required in **`security.jwt.audience`**.
Required OAuth scope is known.	Required in **`security.jwt.required-scope`**. Scope Siebel AI Connectors should match with what is defined for Open Integration.
JWKS URI is available.	Required in **`security.jwt.jwks-uri`**.
Open Integration authentication is aligned.	If Siebel AI Connectors uses OAuth/JWT authentication, Open Integration must also be configured to accept the corresponding authentication flow.
Test token can be generated.	Required for secured endpoint validation.

High-Level Configuration Overview

To ensure that you can start your MCP Servers using Siebel AI Connectors, the following diagram provides a high level guidance on the activities you need to perform when working with the product:

1. **Review Open Integration API(s):** Open Integration is the source of all MCP tools available in `FINAL_SPECIFICATION.json` for Siebel AI Connectors. Begin by reviewing the OpenAPI specification generated by Open Integration, for the target resource & identify the REST operations that support your AI use case. Rather than exposing every available operation, select only the APIs that an AI agent needs to perform its tasks. This helps minimize unnecessary access and simplifies tool management.
2. **Define MCP Tool(s):** After identifying the required operations, move relevant APIs into the `FINAL_SPECIFICATION.json` and create MCP tool definitions that describe how those APIs will be presented to AI agents. Each MCP tool represents a specific business capability, such as retrieving account information or updating a service request. The tool definitions become the catalog of capabilities that can later be assigned to MCP Servers.
3. **Add Open Integration instance(s):** Ensure that the `OI_CONFIG.json` is updated with relevant Open Integration instances which have relevant APIs updated as part of the `FINAL_SPECIFICATION.json` file.
4. **Organize Tools into MCP Server(s):** Group related MCP tools into one or more MCP Servers in the `MCP_CONFIG.json` file. A server can represent a business domain, application area, or functional capability. For example, one server might expose customer-related tools while another exposes product-related operations. Organizing tools into logical servers makes them easier for AI agents to discover and consume.
5. **Configure the Runtime:** Once the tools and servers have been defined, configure the remaining configuration files - `application.yaml` & `logging.properties` with appropriate details. The runtime uses these configured files to establish connections with the configured Open Integration instances, initializes the MCP Servers, and exposes the corresponding endpoints for AI clients.
6. **Deploy and Validate:** Deploy the configured runtime to your target environment, either on-premises or using Siebel Cloud Manager.After deployment, verify that:
    - The runtime starts successfully.
    - The MCP Server endpoints are available.
    - AI clients can discover the configured servers.
    - The exposed tools successfully invoke the corresponding Open Integration REST APIs.

Although creating an MCP Server is primarily a configuration activity, the runtime uses several configuration artifacts to describe the tools, servers, runtime settings, Open Integration connections, and logging behavior. These artifacts work together to define what capabilities are exposed and how the runtime operates. Each artifact has a specific purpose and should be configured consistently before deployment.

Inspecting Open Integration

Open Integration is the source for APIs exposed through Siebel AI Connectors. After Open Integration is configured and running, the resources defined for API generation under `config.json` are exposed as standard REST APIs. The corresponding OpenAPI specification from the Open Integration runtime provides metadata for the exposed resources, including paths, HTTP methods, parameters, request bodies, and schemas.

Use these available Open Integration APIs as the starting point for tool selection. Not every Open Integration operation needs to be exposed as an MCP tool. Select only the operations required for the AI-agent use case.

To prepare MCP tools:

1. Confirm that the required Open Integration resources are configured for API generation under `config.json`.
2. Start or verify the Open Integration runtime.
3. Use the `describe` operation available for the target Siebel artifact configured on Open Integration, to fetch and review the corresponding OpenAPI specification.

4. Identify the API paths and HTTP methods that should be exposed to AI agents.
5. Define the selected operations and MCP tool metadata in `FINAL_SPECIFICATION.json`.
6. Group the selected tools into MCP Servers using `MCP_CONFIG.json`.

The following example shows an Open Integration Account resource exposed as REST API operations:

```
{
 "paths": {
 "/openintegration/v1.0/data/Account/Account/": {
 "get": {
 "operationId": "openintegration_v1.0_data_Account_Account__get"
 },
 "put": {
 "operationId": "openintegration_v1.0_data_Account_Account__put"
 },
 "post": {
 "operationId": "openintegration_v1.0_data_Account_Account__post"
 }
 }
 }
}
```

In this example, the Account resource exposes `get`, `put`, and `post` operations. The implementation team can evaluate these operations and decide which ones should be selectively created as MCP tools.

Open Integration Operation	Expose as MCP Tool?	Guidance
GET /openintegration/v1.0/data/Account/Account/	Common	Expose when the AI-agent use case requires account lookup or retrieval.
PUT /openintegration/v1.0/data/Account/Account/	Conditional	Expose only if the AI-agent use case is allowed to update account data.
POST /openintegration/v1.0/data/Account/Account/	Conditional	Expose only if the AI-agent use case is allowed to create account data.

After the tool selection is complete, define the selected operations in `FINAL_SPECIFICATION.json` and map them to MCP Server endpoints using `MCP_CONFIG.json`.

Configuring Siebel AI Connectors Files

Only the configuration files should be updated. Do not modify `mcp-runtime-<version>.jar` or files under `libs/`. The below table captures details on individual configuration files available for developers to make necessary changes to configure MCP Tools & Servers:

File or Folder	Purpose	Update Guidance
FINAL_SPECIFICATION.json	Cumulated tools in OpenAPI-compatible format. Defines MCP tool metadata and Open Integration operation mappings.	Generated or updated by configuration-time tooling. Do not manually update unless instructed.
MCP_CONFIG.json	MCP Server configuration. Maps MCP Servers to tool names from `FINAL_SPECIFICATION.json`.	Update when MCP Server grouping or endpoint mapping changes.
application.yaml	Runtime configuration. Defines server host, port, security settings, and paths to runtime configuration files.	Update when runtime settings, security settings, or file locations change.

File or Folder	Purpose	Update Guidance
OI_CONFIG.json	Open Integration instance configuration. Defines Open Integration servers used by the runtime.	Update when Open Integration server details change.
logging.properties	Runtime logging configuration.	Update when logging levels or logging behavior need to change.

FINAL_SPECIFICATION.json

`FINAL_SPECIFICATION.json` is the runtime-consumable tool specification used by Siebel AI Connectors to create MCP tool definitions from selected Open Integration REST API operations. It is modeled on the OpenAPI Specification and allows developers to define:

- Open Integration API paths and HTTP methods selected for exposure.
- List of tool names available as part of the FINAL_SPECIFICATION.json
- Tool metadata - tool name, descriptions and so on are defined for selected Open Integration paths and HTTP operations.
- Open Integration server mappings.
- Parameters, request bodies, and schemas used to build and validate MCP tool inputs.

The delivered template uses the following top-level structure:

```
{
 "openapi": "<version>",
 "info": {
 "title": "<Title>",
 "description": "<Description>",
 "contact": {
 "email": "sample@sample.com"
 },
 "version": "<version>"
 },
 "externalDocs": {
 "description": "<description>",
 "url": "<url>"
 },
 "servers": [
 {
 "url": "<Server Base Url>",
 "x-server-name": "<Server Instance Name>"
 }
 ],
 "x-tools": [
 "<Tool Name>"
 ],
 "tags": [
 {
 "name": "<tag>"
 }
 ],
 "paths": {
 "<Path>": {
 "<http-method>": {
 "x-tool-name": "<Tool Name>",
 "x-server-name": "<Server Instance Name>",
 "tags": [
 "<tag>"
 ],
```

```
"operationId": "<Operation Id>",
"description": "<Description>",
"summary": "<Summary>",
"parameters": [],
"requestBody": {},
"responses": {}
}
}
},
"components": {
"schemas": {}
},
"x-original-swagger-version": "3.0"
}
```

Top-Level Fields

Field	Required	Description
components	Recommended	Contains schemas referenced by parameters or request bodies.
externalDocs	Optional	External documentation reference.
info	Recommended	OpenAPI metadata, including title, description, contact, and version.
openapi	Yes	OpenAPI version used by the specification.
paths	Yes	Contains Open Integration API paths and operations selected for tool generation.
servers	Yes	Server entries from the OpenAPI specification. Runtime reads these for server-name and URL comparison or logging. The downstream base URL is resolved from `OI_CONFIG.json`.
tags	Optional	OpenAPI tags used for grouping operations.
x-original-swagger-version	Optional	Preserves source Swagger/OpenAPI version metadata.
x-tools	Yes	List of all tools present in the file.

Additional Details on Fields for FINAL_SPECIFICATION.json

The following describes additional information on the different fields described in the above table, in scope of the FINAL_SPECIFICATION.json

1. servers

   The `servers` array contains OpenAPI server definitions and logical server names.

```
"servers": [
 {
 "url": "<Server Base Url>",
 "x-server-name": "<Server Instance Name>"
 }
```

```
]
```

Field	Required	Description
url	Yes	Server base URL from the OpenAPI specification.
x-server-name	Yes	Logical server instance name. Operation-level **x-server-name** values use this name to identify the Open Integration server mapping.

The runtime reads top-level **servers** for server-name and URL comparison or logging. It does not use the top-level server URL directly as the downstream base URL. The downstream server URL is resolved from **OI_CONFIG.json**.

2. x-tools

   The **x-tools** field lists all tool names present in **FINAL_SPECIFICATION.json**. Each operation-level **x-tool-name** should be present in the top-level **x-tools** array.

```
"x-tools": [
 "Account GET",
 "Account PUT",
 "Account POST"
]
```

3. paths

   The **paths** object contains the selected Open Integration API paths and HTTP method that can be realized as MCP Tools by the Siebel AI Connectors runtime.

Sample:

```
"paths": {
 "/openintegration/v1.0/data/Account/Account/": {
 "get": {
 .....
 },
 "put": {
 .....
 }
 }
}
```

Field	Required	Description
Path key	Yes	OpenAPI path, for example **/accounts/{id}** or **/openintegration/v1.0/data/Account/Account/**.
HTTP method	Yes	HTTP operation under the path, such as **get**, **post**, **put**, or **delete**.

4. HTTP method Fields

For Siebel AI Connectors, we map individual HTTP methods(GET, PUT, POST and DELETE) defined under a "`path`" to an MCP Tool in the FINAL_SPECIFICATION.json file. Each selected HTTP method defines the OpenAPI operation and MCP-specific metadata. The following table captures relevant information on the different key-value pairs that should be configured:

Field	Required	Description
x-tool-name	Recommended	Primary source for the MCP tool name. If absent or blank, **`operationId`** is used as the fallback tool name.
x-server-name	Optional	Identifies which server entry from **`OI_CONFIG.json`** the tool sends the downstream request to. If missing, the default server from **`OI_CONFIG.json`** is used.
operationId	Yes	Used as fallback MCP tool name when **`x-tool-name`** is absent or blank. Also used in description fallback behavior.
description	Recommended	Primary source for the MCP tool description. Can be updated by user to reflect the tool description they would want to provide for the selected operation.
summary	Optional	Fallback MCP tool description when **`description`** is absent or blank.
tags	Optional	Provided by the Open Integration OpenAPI Spec.
parameters	Conditional	Provided by the Open Integration OpenAPI Spec.
requestBody	Conditional	Provided by the Open Integration OpenAPI Spec. Required when the operation expects a request body. Used to build the body argument in the MCP input schema.
responses	Recommended	Provided by the Open Integration OpenAPI Spec.

5. parameters

Parameters are available for certain HTTP methods available for a given `path`. These parameters are specified in Open Integration's OpenAPI spec and are added as needed in the **`FINAL_SPECIFICATION.json`** for the associated path and HTTP method.

`Sample:`

```
"parameters": [
 {
```

```
 "name": "Id",
 "in": "query",
 "description": "Account identifier.",
 "schema": {
 "type": "string"
 }
 }
]
```

Field	Required	Description
in	Yes	Provided via the Open Integration OpenAPI Spec
name	Yes	Provided via the Open Integration OpenAPI Spec
schema	Yes	Provided via the Open Integration OpenAPI Spec
description	Recommended	Optionally available. Should be added for the parameter for proper usage with the MCP tool.

6. components.schemas

The `components.schemas` section defines schemas used by parameters and request bodies.

Sample:

```
"components": {
 "schemas": {
 "AccountUpdate": {
 "type": "object",
 "additionalProperties": false,
 "required": [
 "Name"
 ],
 "properties": {
 "Name": {
 "type": "string",
 "description": "Account name."
 }
 }
 }
 }
}
```

Field	Description
required	Validated for request body. This is distinct from the generated top-level tool required array.
additionalProperties	Controls whether unexpected body fields are allowed in a request body. Can be set to true or false. If set to `false`, unexpected body fields are rejected.

Field	Description
	If set to true, additional fields are allowed as part of the Request Body.
description	Exposed to clients. Body descriptions appear on `properties.body`.
discriminator	Preserved. Runtime has no discriminator-specific logic, although schema composition may apply where supported.

MCP_CONFIG.json

`MCP_CONFIG.json` defines MCP Servers and maps each MCP Server to tool names from `FINAL_SPECIFICATION.json`.

Sample:

```
{
 "mcp-servers": [
 {
 "mcp-server-name": "GET",
 "mcp-server-path": "/account1",
 "mcp-server-description": "Account 1 Description",
 "mcp-server-version": "1.0.0",
 "mcp-server-tools": [
 "Account GET",
 "Account PUT",
 "Account POST"
 ]
 },
 {
 "mcp-server-name": "PUT",
 "mcp-server-path": "/account2",
 "mcp-server-description": "Account 2 Description",
 "mcp-server-version": "1.0.0",
 "mcp-server-tools": [
 "Account GET",
 "AccountMethod POST",
 "AccountKeyMethod POST"
 ]
 }
 ]
}
```

Field	Required	Description
mcp-servers	Yes	Array of MCP Server definitions.
mcp-server-name	Yes	Name of the MCP Server.
mcp-server-path	Yes	Runtime endpoint path for the MCP Server. Must be unique for each MCP Server.
mcp-server-description	Yes	Description of the MCP Server.
mcp-server-version	Yes	MCP Server version.

Field	Required	Description
mcp-server-tools	Yes	List of tool names exposed by this MCP Server. Each value should be present under x-tools in `FINAL_SPECIFICATION.json`

Rules:

- Each `mcp-server-path` must be unique.
- Each tool in `mcp-server-tools` must exist in `FINAL_SPECIFICATION.json`.
- Tool names are matched by exact string.

OI_CONFIG.json

`OI_CONFIG.json` defines Open Integration instance(s) used by the runtime.

Sample:

```
{
 "default-server": "<server-name-1>",
 "servers": [
 {
 "name": "<server-name-1>",
 "scheme": "<scheme>",
 "host": "<host>",
 "port": 8433,
 "context": "<context>"
 },
 {
 "name": "<server-name-2>",
 "scheme": "<scheme>",
 "host": "<host>",
 "port": 8433,
 "context": "<context>"
 }
 ]
}
```

Field	Required	Description
default-server	Yes	Default Open Integration server. Used when operation-level **`x-server-name`** is missing or unknown.
servers	Yes	List of Open Integration server definitions.
name	Yes	Logical Open Integration server name. This should match operation-level **`x-server-name`** values in **`FINAL_SPECIFICATION.json`**.
scheme	Yes	Protocol used for Open Integration, such as **`http`** or **`https`**.
host	Yes	Open Integration host.
port	Yes	Open Integration port.
context	Yes	Open Integration context path.

Rules:

- `default-server` must match one of the `servers[].name` values.
- Each `servers[].name` must be unique.
- Operation-level `x-server-name` values in `FINAL_SPECIFICATION.json` should resolve to entries in `OI_CONFIG.json`.
- If operation-level `x-server-name` is missing or unknown, the runtime uses `default-server`.

logging.properties

The `logging.properties` defines runtime logging behavior and must be referenced from `application.yaml` using `mcp.logging-properties-file`.

The file should be placed in the same folder as `mcp-runtime-<version>.jar` unless the path in `application.yaml` is updated.

```
handlers=io.helidon.logging.jul.HelidonConsoleHandler
java.util.logging.SimpleFormatter.format=%1$tY.%1$tm.%1$td %1$tH:%1$tM:%1$tS.%1$tL %4$s %5$s%6$s%n

# Global logging level. Can be overridden by specific loggers
.level=SEVERE
io.helidon.level=SEVERE
```

Field	Description
handlers	Defines the logging handler used by the runtime.
java.util.logging.SimpleFormatter.format	Defines the log message format.
.level	Global logging level. Can be overridden by specific loggers.
io.helidon.level	Logging level for Helidon runtime loggers.

application.yaml

`application.yaml` defines runtime server settings, JWT security settings, and configuration file paths.

```
Sample:

server:
 port: <port>
 host: <host>

security:
 jwt:
 enabled: true
 issuer: <issuer>
 audience: <audience>
 required-scope: <scope>
 clock-skew-seconds: <seconds>
 jwks-uri: "<jwks-uri>"
 jwks-cache-duration-seconds: <duration>

mcp:
 server-config-file: <path to MCP_CONFIG.json>
 final-spec-file: <path to FINAL_SPECIFICATION.json>
 logging-properties-file: <path to logging.properties>
 open-int-config-file: <path to OI_CONFIG.json>
```

Field	Required	Description
server.port	Yes	Runtime server port. Change this if the configured port is already in use.
server.host	Yes	Runtime server host.
security.jwt.enabled	Required when JWT is used	Enables or disables JWT validation.
security.jwt.issuer	Required when JWT is used	Expected token issuer.
security.jwt.audience	Required when JWT is used	Expected token audience.
security.jwt.required-scope	Required when JWT is used	Required OAuth scope for MCP requests.
security.jwt.clock-skew-seconds	Recommended when JWT is used	Allowed clock skew during token validation.
security.jwt.jwks-uri	Required when JWT is used	JWKS URI used to validate JWT signatures.
security.jwt.jwks-cache-duration-seconds	Recommended when JWT is used	Duration for JWKS cache.
mcp.server-config-file	Yes	Path to `MCP_CONFIG.json`.
mcp.final-spec-file	Yes	Path to `FINAL_SPECIFICATION.json`.
mcp.logging-properties-file	Yes	Path to `logging.properties`.
mcp.open-int-config-file	Yes	Path to `OI_CONFIG.json`.

Security and Authentication

Siebel AI Connectors support OAuth 2.0 JWT-based authentication. When JWT authentication is enabled, each request to an MCP Server endpoint must include a valid access token.

The runtime validates the token using the JWT configuration in `application.yaml`.

```
security:
 jwt:
 enabled: true
 issuer: <issuer>
 audience: <audience>
 required-scope: <scope>
 clock-skew-seconds: <seconds>
 jwks-uri: "<jwks-uri>"
 jwks-cache-duration-seconds: <duration>
```

Field	Required	Description
security.jwt.enabled	Yes	Enables JWT authentication.
security.jwt.issuer	Yes	Expected token issuer.
security.jwt.audience	Yes	Expected token audience.
security.jwt.required-scope	Yes	OAuth scope required to invoke MCP endpoints.
security.jwt.clock-skew-seconds	Recommended	Allowed clock skew during token validation.
security.jwt.jwks-uri	Yes	JWKS endpoint used to validate token signatures.

Field	Required	Description
security.jwt.jwks-cache-duration-seconds	Recommended	Duration for JWKS cache.

In a secured deployment:

1. The MCP client generates an OAuth access token from the configured identity provider.
2. The MCP client sends the request to the MCP Server endpoint with an authorization header.
3. The Siebel AI Connectors runtime validates the JWT.
4. After validation, the runtime invokes the configured Open Integration REST API.

Note: Authorization: Bearer <access_token>

The authentication configuration must be consistent across Siebel AI Connectors and Open Integration. If Siebel AI Connectors uses OAuth/JWT authentication, Open Integration must also be configured to accept the corresponding authentication flow.

Consistency Checklist

Before starting or deploying the runtime, verify the following:

Check	Required
Every tool in **`MCP_CONFIG.json`** has a mapping in **`FINAL_SPECIFICATION.json`**.	Yes
Every operation-level **`x-tool-name`** is listed in top-level **`x-tools`**.	Yes
`mcp.server-config-file` points to **`MCP_CONFIG.json`**.	Yes
`mcp.final-spec-file` points to **`FINAL_SPECIFICATION.json`**.	Yes
`mcp.logging-properties-file` points to **`logging.properties`**.	Yes
`mcp.open-int-config-file` points to **`OI_CONFIG.json`**.	Yes
Every operation-level **`x-server-name`** resolves to an entry in **`OI_CONFIG.json`**, or **`default-server`** is configured for fallback.	Yes
`OI_CONFIG.json` defines a valid **`default-server`**.	Yes
Each **`mcp-server-path`** is unique.	Yes
Request bodies use **`application/json`**.	Yes
Supported parameter locations are **`path`**, **`query`**, and **`header`**.	Yes

Deploying Siebel AI Connectors

This section describes how to deploy Siebel AI Connectors after the required configuration files are prepared.

Siebel AI Connectors can be deployed in an on-premises environment or through Siebel Cloud Manager. In both deployment models, the runtime uses the configured artifacts to create MCP Servers, expose MCP-compatible tools, and connect those tools to the selected Siebel Open Integration REST APIs.

Before deploying Siebel AI Connectors, verify that `application.yaml`, `MCP_CONFIG.json`, `FINAL_SPECIFICATION.json`, `OI_CONFIG.json`, and `logging.properties` are updated for the target environment. Also verify that Siebel Open Integration is installed, configured, and running.

The deployment topics include:

- Deploying Siebel AI Connectors - On-Premise
- Deploying Siebel AI Connectors - Siebel Cloud Manager
- Runtime Behavior and Validations

Deployment Prerequisites

Java and Certificate Prerequisites

For on-premises runtime startup, verify the following:

Prerequisite	Description
JRE 21 is available.	Configure `JAVA_HOME` and `PATH` for the JRE delivered or supported with the Siebel installer.
Required certificates are trusted.	If Open Integration uses TLS, import the required Siebel or Open Integration certificate into the Java truststore used by the runtime.
Truststore path and password are known.	Required when configuring Java truststore behavior for deployment or runtime.

Example certificate import command on Linux:

```
keytool -importcert -trustcacerts -alias siebel \
 -file "<FS path>/siebel.crt" \
 -keystore "<FS path>/jre/lib/security/cacerts" \
 -storepass changeit \
 -noprompt
```

Example certificate import command on Windows:

```
keytool -importcert -trustcacerts -alias siebel ^
 -file "<FS path>\siebel.crt" ^
 -keystore "<FS path>\jre\lib\security\cacerts" ^
 -storepass changeit ^
 -noprompt
```

SCM Deployment Prerequisites

For Siebel Cloud Manager deployment, verify the following before deploying Siebel AI Connectors:

Prerequisite	Description
Siebel Cloud Manager is available.	SCM APIs are used to create infrastructure and deploy Siebel AI Connectors.

Prerequisite	Description
Kubernetes environment is available.	For example, BYO OCNE, OKE, or another supported Kubernetes target.
Infrastructure input is available.	Namespace, kubeconfig path or cluster details, operator options, registry details, and GitOps details are required.
Customer Git repository is available.	Runtime configuration artifacts are maintained in Git.
Git access credentials are available.	HTTPS token, self-signed CA path, or SSH private key path may be required.
Git repository is registered with SCM.	SCM returns a `git_id` used in the deployment payload.
Container registry access is available.	Registry URL, user, password or token, and registry prefix are required.
Configuration files are committed to Git.	`application.yaml`, `MCP_CONFIG.json`, `FINAL_SPECIFICATION.json`, `OI_CONFIG.json`, and `logging.properties` must be available in the configured Git folder.
Truststore file is available when required.	The deployment payload can reference the Java truststore path and password.
Namespace is created when using BYO namespace.	If `byo_ns` is `true`, create the namespace before deployment and use the namespace value in the deployment `name` field.
Security context values are known when required.	If a custom security context is used, provide `run_as_user`, `run_as_group`, and `fs_group`.

Deploying Siebel AI Connectors - On-Premises

This topic describes how to deploy Siebel AI Connectors in an on-premises environment after the required configuration files are prepared.

In an on-premises deployment, Siebel AI Connectors are started from the delivered runtime package. Before starting the runtime, verify that the configuration files are present, that the Open Integration server details are correct, and that the Java prerequisites are complete.

Before You Begin

Before deploying Siebel AI Connectors on-premise, make sure that:

- Siebel Open Integration is installed, configured, and running.
- The Open Integration REST APIs selected for MCP tool exposure are available.
- The Siebel AI Connectors runtime package is installed.
- The required configuration files are updated for the target environment.
  - Navigate to the Siebel AI Connectors runtime directory and verify that the necessary files, libs and folders are present.
  - Verify that `application.yaml` contains the correct runtime settings and file paths.
    - `server.port` is available on the target host.
    - `server.host` is correct for the deployment.
    - The paths under `mcp` point to the correct files.
    - JWT settings are correct, if authentication is enabled.
  - Verify that `OI_CONFIG.json` contains the Open Integration server instances used by the runtime.
    - `default-server` matches one of the server names in the `servers` array.
    - Each server name is unique.
    - `scheme`, `host`, `port`, and `context` are correct for the Open Integration environment.
    - The runtime host can connect to each configured Open Integration server.

    - Verify that `FINAL_SPECIFICATION.json` contains the selected Open Integration operations and MCP tool metadata.
        - The required tools are present.
        - Tool names are unique.
        - Tool names used in `MCP_CONFIG.json` match the names in `FINAL_SPECIFICATION.json`.
        - The Open Integration server mappings are correct.
    - Verify that `MCP_CONFIG.json` defines the MCP Servers and maps each server to one or more tools from `FINAL_SPECIFICATION.json`.
        - Each `mcp-server-path` is unique.
        - Each tool listed in `mcp-server-tools` exists in `FINAL_SPECIFICATION.json`.
        - Tool names match exactly between `MCP_CONFIG.json` and `FINAL_SPECIFICATION.json`.
- JRE 21 is configured.
- Required certificates are imported into the Java truststore, if Open Integration uses TLS.

Start the Siebel AI Connectors Runtime

After verifying the configuration files and necessary prerequisites, start the runtime from the runtime directory.

For Windows, run:

```
java -cp ".\mcp-runtime-<version>.jar;.\libs\*" com.oracle.siebel.mcp.Main
```

For Linux, run:

```
java -cp "./mcp-runtime-<version>.jar:./libs/*" com.oracle.siebel.mcp.Main
```

Replace `<version>` with the version of the runtime JAR delivered in the package.

Verify Runtime Startup

After starting the runtime, review the startup logs.

Look for log messages that confirm the following:

- `FINAL_SPECIFICATION.json` was parsed successfully.
- Open Integration servers were loaded.
- The default Open Integration server was identified.
- Tools were processed from the OpenAPI specification.
- The runtime selected the Open Integration server for each tool.
- MCP tools were adapted from the OpenAPI specification.
- MCP Servers were started successfully.
- The base MCP Server URL was created.

Example log messages:

```
INFO Successfully parsed OpenAPI spec file:<Path>/FINAL_SPECIFICATION.json

INFO Loaded Open Integration servers: [OI123, OI456]; default server: OI456

INFO Processing Tool : [Account GET] , Method : [GET] , Path: [/openintegration/v1.0/data/Account/Account/]

INFO Using Open Integration server [OI456] for tool [Account GET]
```

```
INFO Adapted [20] tools from OpenAPI spec

INFO Started all channels in 8 milliseconds. 590 milliseconds since JVM startup. Java 21.0.11

INFO MCP Servers started with base url: 0.0.0.0:8081/siebelAIConnectors/mcp
```

If the runtime does not start successfully, review the logs and verify the configuration file paths, Open Integration server details, Java configuration, certificate configuration, and MCP tool mappings.

Verify MCP Server Access

After the Siebel AI Connectors runtime starts successfully, verify that the configured MCP Servers are available.

The runtime exposes MCP Servers using the base MCP path and the MCP Server path configured in **MCP_CONFIG.json**.

The MCP Server endpoint uses the following format:

```
<scheme>://<host>:<port>/siebelAIConnectors/mcp/<mcp-server-path>
```

Where:

Value	Description
<scheme>	The protocol used to access the Siebel AI Connectors runtime, such as **http** or **https**.
<host>	The host where the Siebel AI Connectors runtime is running.
<port>	The runtime port configured in **application.yaml**.
/siebelAIConnectors/mcp	The base path used by the Siebel AI Connectors runtime for MCP Servers.
<mcp-server-path>	The MCP Server path configured in **MCP_CONFIG.json**.

For example, if **application.yaml** configures the runtime on port **8081**, and **MCP_CONFIG.json** defines the following MCP Server path:

```
{
 "mcp-server-name": "Account MCP Server",
 "mcp-server-path": "/account",
 "mcp-server-description": "MCP Server for account-related Siebel operations.",
 "mcp-server-version": "1.0.0",
 "mcp-server-tools": [
 "get_account_details",
 "update_account_details"
 ]
}
```

Then the MCP Server endpoint is available at:

```
http://<host>:8081/siebelAIConnectors/mcp/account
```

If multiple MCP Servers are configured, each server is available at its own configured path.

Example:

MCP Server Name	MCP Server Path	MCP Server Endpoint
Account MCP Server	/account	http://<host>:8081/siebelAIConnectors/mcp/account

MCP Server Name	MCP Server Path	MCP Server Endpoint
Product MCP Server	/product	http://<host>:8081/siebelAIConnectors/mcp/product
Service Request MCP Server	/service-request	http://<host>:8081/siebelAIConnectors/mcp/service-request

Use an MCP client, such as Postman, MCP Inspector or another supported MCP client, to connect to the required MCP Server endpoint and verify that the configured tools are available.

If the MCP Server endpoint is not available, verify the following:

- The Siebel AI Connectors runtime started successfully.
- The runtime startup logs show the MCP Server base URL.
- The `server.port` value in `application.yaml` is correct.
- The `mcp-server-path` value in `MCP_CONFIG.json` is correct.
- Each `mcp-server-path` value is unique.
- The MCP Server path in the client request matches the value configured in `MCP_CONFIG.json`.
- Network access to the runtime host and port is available.
- Authentication headers are included if JWT authentication is enabled.

Updating an Existing On-Premises Deployment

The runtime does not dynamically reload configuration changes.

- If `FINAL_SPECIFICATION.json` is updated, review and update the tool mappings in `MCP_CONFIG.json`, and then restart the runtime.
- If `MCP_CONFIG.json` is updated, restart the runtime.
- If `application.yaml` or `OI_CONFIG.json` is updated, restart the runtime.

Deploying Siebel AI Connectors using Siebel Cloud Manager

This section describes the Siebel Cloud Manager payloads used to deploy Siebel AI Connectors in a Kubernetes-based environment.

Siebel AI Connectors expose selected Siebel Open Integration REST APIs as MCP-compatible tools. In an SCM deployment, the runtime configuration artifacts are maintained in a customer Git repository and deployed through Siebel Cloud Manager.

The deployment process includes the following high-level steps:

1. Retrieve the Siebel AI Connectors configuration templates.
2. Register the customer Git repository.
3. Create the required infrastructure.
4. Verify the infrastructure.
5. Deploy Siebel AI Connectors.
6. Check the deployment status.
7. Delete the deployment, if required.
8. Perform incremental updates after configuration changes.

1. Retrieve Configuration Templates

Use this API to retrieve the base configuration templates required for Siebel AI Connectors.

```
API
GET /scm/api/v1.0/siebelAIConnectors/templates HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>
```

The response contains a ZIP file with the following template artifacts:

```
Expected Template Artifacts:
application.yaml, MCP_CONFIG.json, OI_CONFIG.json, FINAL_SPECIFICATION.json, logging.properties
```

These files are updated by the customer and committed to the Git repository used by the Siebel AI Connectors deployment.

2. Register Customer Git Repository

Use this API to register the Git repository that contains the Siebel AI Connectors configuration artifacts.

SCM validates access to the repository and returns a `git_id`. This `git_id` is used later in the Siebel AI Connectors deployment payload.

```
API
POST /scm/api/v1.0/git_repositories HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>
Content-Type: application/json
```

```
API
POST /scm/api/v1.0/git_repositories HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>
Content-Type: application/json

Sample HTTPS Git Payload with Self-Signed CA
{
 "git": {
 "git_protocol_type": "https",
 "git_url": "https://<git_host>/<group>/<repo>.git",
 "git_user": "<git_user>",
 "git_accesstoken": "<git_access_token>",
 "git_selfsigned_cacert": "/home/opc/<path_to_ca_certificate>"
 }
}

Sample SSH Git Payload
{
 "git": {
 "git_protocol_type": "ssh",
 "git_url": "git@<git_host>:<group>/<repo>.git",
 "git_ssh_private_key": "/home/opc/.ssh/<private_key_file>"
 }
}

Example Response
{
 "data": {
 "git_id": "<git_id>",
 "git_url": "https://<git_host>/<group>/<repo>.git",
 "git_protocol_type": "https"
 },
 "message": "Git repository registered successfully.",
 "status": "success"
}
```

3. Create Infrastructure

Use this API to create the infrastructure required for the Siebel AI Connectors deployment.

The payload contains the following top-level sections:

Section	Description
operators	Defines operator installation requirements.
kubernetes	Defines the Kubernetes environment where the deployment will run.
git	Defines the GitOps and Helm chart repositories used by SCM.
registry	Defines the container registry used for images.
security_context	Optional. Defines custom non-root user and group IDs.

```
API
POST /scm/api/v1.0/infrastructure HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>
Content-Type: application/json

Sample Infrastructure Payload without Security Context
{
 "operators": [
 {
 "coherence_operator":< false | true >
 }
 ],
 "kubernetes": {
 "namespace": "<infra_namespace>",
 "kubernetes_type": "BYO_OCNE",
 "byo_ocne": {
 "kubeconfig_path": "/home/opc/siebel/<kubeconfig_file>.yaml"
 }
 },
 "git": {
 "git_type": "byo_git",
 "byo_git": {
 "git_protocol_type": "https",
 "git_accesstoken": "<git_access_token>",
 "git_user": "<git_user>",
 "git_scm_repo_url": "https://<git_host>/<group>/<siebel_ai_connectors_gitops_repo>",
 "git_scm_repo_branch": "main",
 "git_scm_flux_folder": "flux",
 "git_helm_repo_url": "https://<git_host>/<group>/<helm_chart_repo>",
 "git_helm_repo_branch": "main",
 "skip_flux_bootstrap": false
 }
 },
 "registry": {
 "registry_url": "<registry_host>",
 "registry_user": "<registry_user>",
 "registry_password": "<registry_password>",
 "registry_prefix": "<registry_prefix>"
 }
}
```

Use the following payload when infrastructure and operator setup must use custom non-root IDs:

```
Sample Infrastructure Payload with Security Context
```

```
{
 "security_context": {
 "run_as_user": <uid>,
 "run_as_group": <gid>,
 "fs_group": <fs_group_id>
 },
 "operators": [
 {
 "coherence_operator": < false | true >
 }
 ],
 "kubernetes": {
 "namespace": "<infra_namespace>",
 "kubernetes_type": "BYO_OCNE",
 "byo_ocne": {
 "kubeconfig_path": "/home/opc/siebel/<kubeconfig_file>.yaml"
 }
 },
 "git": {
 "git_type": "byo_git",
 "byo_git": {
 "git_protocol_type": "https",
 "git_accesstoken": "<git_access_token>",
 "git_user": "<git_user>",
 "git_scm_repo_url": "https://<git_host>/<group>/<siebel_ai_connectors_gitops_repo>",
 "git_scm_repo_branch": "main",
 "git_scm_flux_folder": "flux",
 "git_helm_repo_url": "https://<git_host>/<group>/<helm_chart_repo>",
 "git_helm_repo_branch": "main",
 "skip_flux_bootstrap": false
 }
 },
 "registry": {
 "registry_url": "<registry_host>",
 "registry_user": "<registry_user>",
 "registry_password": "<registry_password>",
 "registry_prefix": "<registry_prefix>"
 }
}
```

Sample Response

```
{
 "data": {
 "infra_details": {
 "components_info": {},
 "infra_id": "<infra_id>",
 "infra_status": "creation-in-progress",
 "payload": {
 "git": {
 "byo_git": {
 "git_accesstoken": "********",
 "git_helm_repo_branch": "main",
 "git_helm_repo_url": "https://<git_host>/<group>/<helm_chart_repo>",
 "git_protocol_type": "http",
 "git_scm_flux_folder": "flux",
 "git_scm_repo_branch": "main",
 "git_scm_repo_url": "https://<git_host>/<group>/<siebel_ai_connectors_gitops_repo>",
 "git_user": "<git_user>",
 "skip_flux_bootstrap": false
 },
 "git_type": "byo_git"
 },
 "kubernetes": {
 "byo_ocne": {
 "kubeconfig_path": "/home/opc/siebel/kubeconfig.yaml"
```

```
 },
 "kubernetes_type": "BYO_OCNE",
 "namespace": "<infra_namespace>"
 },
 "operators": [
 {
 "coherence_operator": true
 }
 ],
 "registry": {
 "registry_password": "************",
 "registry_prefix": "<registry_prefix>",
 "registry_url": "<registry_host>",
 "registry_user": "<registry_user>"
 }
 },
 "resource_status": {
 "git": "not-validated",
 "kubernetes": "not-validated",
 "operators": [
 {
 "coherence_operator": "not-validated"
 }
 ],
 "registry": "not-validated"
 },
 "self_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>",
 "stages_info": [
 {
 "end_time": "",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
setup_validation_helmchart",
 "log_location": "",
 "name": "Setup Helm Chart for validation",
 "previous_status": "",
 "stage_name": "setup_validation_helmchart",
 "start_time": "<start_time>",
 "status": ""
 },
 {
 "end_time": "",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
check_job_status_helmchart",
 "log_location": "",
 "name": "Check Job status",
 "previous_status": "",
 "stage_name": "check_job_status_helmchart",
 "start_time": "<start_time>",
 "status": ""
 },
 {
 "end_time": "",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
install_operator_workflow",
 "log_location": "",
 "name": "Install operator workflow",
 "previous_status": "",
 "stage_name": "install_operator_workflow",
 "start_time": "<start_time>",
 "status": ""
 },
 {
 "end_time": "",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
verify_operator_workflow",
 "log_location": "",
```

```
 "name": "Verify Operator Installation",
 "previous_status": "",
 "stage_name": "verify_operator_workflow",
 "start_time": "<start_time>",
 "status": ""
 },
 {
 "end_time": "",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
collect_infra_artifacts",
 "log_location": "",
 "name": "Collect Infra artifacts",
 "previous_status": "",
 "stage_name": "collect_infra_artifacts",
 "start_time": "<start_time>",
 "status": ""
 }
 ]
 }
 },
 "message": "Infrastructure successfully created.",
 "status": "success"
}
```

After this response is returned, use the infra_id value from data.infra_details.infra_id in the subsequent infrastructure verification and Siebel AI Connectors deployment steps.

4. Verify Infrastructure

Use this API to verify that the infrastructure has been created successfully.

```
API
GET /scm/api/v1.0/infrastructure/<infra_id> HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>

Sample Response
{
 "data": {
 "infra_details": {
 "components_info": {
 "coherence_operator": true,
 "git": {
 "byo_git": {
 "git_accesstoken": "*************",
 "git_helm_repo_branch": "<helm_repo_branch>",
 "git_helm_repo_url": "https://<git_host>/<group>/<helm_chart_repo>",
 "git_protocol_type": "<http|https|ssh>",
 "git_scm_flux_folder": "<flux_folder>",
 "git_scm_repo_branch": "<gitops_repo_branch>",
 "git_scm_repo_url": "https://<git_host>/<group>/<gitops_repo>",
 "git_user": "<git_user>",
 "skip_flux_bootstrap": false
 },
 "git_type": "byo_git"
 },
 "kubernetes": {
 "byo_ocne": {
 "kubeconfig_path": "/home/opc/siebel/<kubeconfig_file>.yaml"
 },
 "k8s_profile": "/home/opc/siebel/infrastructures/<infra_id>/k8sprofile",
 "kubeconfig_path": "/home/opc/siebel/infrastructures/<infra_id>/kubeconfig.yaml",
 "kubernetes_type": "BYO_OCNE",
 "namespace": "<infra_namespace>"
 },
 "registry": {
```

```
"registry_password": "************",
"registry_prefix": "<registry_prefix>",
"registry_url": "<registry_host>",
"registry_user": "<registry_user>"
}
},
"infra_id": "<infra_id>",
"infra_status": "completed",
"payload": {
"git": {
"byo_git": {
"git_accesstoken": "**********",
"git_helm_repo_branch": "<helm_repo_branch>",
"git_helm_repo_url": "https://<git_host>/<group>/<helm_chart_repo>",
"git_protocol_type": "<http|https|ssh>",
"git_scm_flux_folder": "<flux_folder>",
"git_scm_repo_branch": "<gitops_repo_branch>",
"git_scm_repo_url": "https://<git_host>/<group>/<gitops_repo>",
"git_user": "<git_user>",
"skip_flux_bootstrap": false
},
"git_type": "byo_git"
},
"kubernetes": {
"byo_ocne": {
"kubeconfig_path": "/home/opc/siebel/<kubeconfig_file>.yaml"
},
"k8s_profile": "/home/opc/siebel/infrastructures/<infra_id>/k8sprofile",
"kubeconfig_path": "/home/opc/siebel/infrastructures/<infra_id>/kubeconfig.yaml",
"kubernetes_type": "BYO_OCNE",
"namespace": "<infra_namespace>"
},
"operators": [
{
"coherence_operator": true
}
],
"registry": {
"registry_password": "*********",
"registry_prefix": "<registry_prefix>",
"registry_url": "<registry_host>",
"registry_user": "<registry_user>"
}
},
"resource_status": {
"git": "validated",
"kubernetes": "validated",
"operators": [
{
"coherence_operator": "validated"
}
],
"registry": "validated"
},
"self_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>",
"stages_info": [
{
"end_time": "<end_time>",
"log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
setup_validation_helmchart",
"log_location": "/home/opc/siebel/infrastructures/<infra_id>/logs/install_connection_chart.log",
"name": "Setup Helm Chart for validation",
"previous_status": "",
"stage_name": "setup_validation_helmchart",
"start_time": "<start_time>",
"status": "passed"
```

```
 },
 {
 "end_time": "<end_time>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
check_job_status_helmchart",
 "log_location": "/home/opc/siebel/infrastructures/<infra_id>/logs/validate_connection_chart.log",
 "name": "Check Job status",
 "previous_status": "",
 "stage_name": "check_job_status_helmchart",
 "start_time": "<start_time>",
 "status": "passed"
 },
 {
 "end_time": "<end_time>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
install_operator_workflow",
 "log_location": "/home/opc/siebel/infrastructures/<infra_id>/logs/operators.log",
 "name": "Install operator workflow",
 "previous_status": "",
 "stage_name": "install_operator_workflow",
 "start_time": "<start_time>",
 "status": "passed"
 },
 {
 "end_time": "<end_time>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
verify_operator_workflow",
 "log_location": "/home/opc/siebel/infrastructures/<infra_id>/logs/operators.log",
 "name": "Verify Operator Installation",
 "previous_status": "",
 "stage_name": "verify_operator_workflow",
 "start_time": "<start_time>",
 "status": "passed"
 },
 {
 "end_time": "<end_time>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/infrastructure/<infra_id>/logs/
collect_infra_artifacts",
 "log_location": "/home/opc/siebel/infrastructures/<infra_id>/logs/artifacts.log",
 "name": "Collect Infra artifacts",
 "previous_status": "",
 "stage_name": "collect_infra_artifacts",
 "start_time": "<start_time>",
 "status": "passed"
 }
 ]
 }
 },
 "message": "Infrastructure details retrieved successfully.",
 "status": "success"
}
```


After verification completes successfully, the `infra_status` value is `completed`, and the `resource_status` values for Git, Kubernetes, operators, and registry are marked as `validated`. Use the `infra_id` value in the Siebel AI Connectors deployment payload.

5. Deploy Siebel AI Connectors

Use this API to deploy Siebel AI Connectors using the prepared infrastructure and registered Git repository. The deployment payload references the Git repository that contains the runtime configuration files.

```
API
POST /scm/api/v1.0/siebelAIConnectors HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>
```

```
Content-Type: application/json

Deployment Payload
{
 "name": "<mcp_deployment_namespace>",
 "infra_id": "<infra_id>",
 "byo_ns": false,
 "security_context": {
 "run_as_user": <uid>,
 "run_as_group": <gid>,
 "fs_group": <fs_group_id>
 },
 "siebel_mcp": {
 "git": {
 "git_id": "<git_id>",
 "git_repo_branch": "main",
 "git_config_folder": "siebel-mcp"
 },
 "java_truststore": {
 "cacerts_path": "/home/opc/siebel/cacerts",
 "password": "<truststore_password>"
 }
 }
}
```

Optionally, If `byo_ns` is set to "`true`", ensure that the Siebel AI Connector namespace is created before executing this step. Then, specify the created namespace in the Siebel AI Connector deployment payload by setting the "`name`" field to the namespace value.

```
Example Response
{
 "data": {
 "ansible_chart_name": "ansible-<deployment_name>-<deploy_id_lowercase>",
 "app_dir": "/home/opc/siebel/microservices/siebel-ai-connectors/<deploy_id>",
 "byo_ns": false,
 "deploy_id": "<deploy_id>",
 "deploy_status": "creation-in-progress",
 "infra_id": "<infra_id>",
 "name": "<mcp_deployment_namespace>",
 "namespace": "<mcp_deployment_namespace>",
 "product_name": "siebel-ai-connectors",
 "security_context": {
 "fs_group": <fs_group_id>,
 "run_as_group": <gid>,
 "run_as_user": <uid>
 },
 "self_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/siebelAIConnectors/<deploy_id>",
 "siebel_mcp": {
 "git": {
 "git_config_folder": "siebel-mcp",
 "git_id": "<git_id>",
 "git_json": {
 "branch": "main",
 "config_folder": "siebel-mcp",
 "protocol_type": "https",
 "url": "https://<git_host>/<group>/<repo>",
 "user": "<git_user>"
 },
 "git_protocol_type": "https",
 "git_repo_branch": "main",
 "git_selfsigned_cacert": "",
 "git_siebel_mcp_directory": "siebel-mcp",
 "git_ssh_private_key": "",
 "git_url": "https://<git_host>/<group>/<repo>",
 "git_user": "<git_user>"
```

```
},
"java_truststore": {
"cacerts_path": "/home/opc/siebel/cacerts",
"helm_values": {
"checksum": "<truststore_checksum>",
"enabled": true,
"fileName": "cacerts",
"mountPath": "/opt/mcp/truststore",
"password": "",
"passwordSecretKey": "storepass",
"passwordSecretName": "",
"secretKey": "cacerts",
"secretName": ""
},
"password": "<masked_or_configured_truststore_password>"
}
},
"stages": [
{
"name": "Pre Deploy",
"stage_name": "pre_deploy",
"stage_description": "Create pre requisites and other preparations",
"status": ""
},
{
"name": "Prepare GitOps",
"stage_name": "prepare_gitops",
"stage_description": "Generate Flux definitions, Prepare helmchart to git and Prepare GitOps",
"status": ""
},
{
"name": "Verify Deployment",
"stage_name": "verify_deployment",
"stage_description": "Verify HR Deployment",
"status": ""
},
{
"name": "Publish Urls",
"stage_name": "publish_urls",
"stage_description": "Publish Application Urls",
"status": ""
}
]
},
"message": "ansible playbook for siebel-ai-connectors ansible playbook for executed successfully",
"status": "success"
}
```


6. Get Siebel AI Connectors Deployment Status

Use this API to retrieve the current status of a Siebel AI Connectors deployment.

```
API
GET /scm/api/v1.0/siebelAIConnectors/<deploy_id> HTTP/1.1
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>

Example Response
```


```
{
"data": {
"ansible_chart_name": "ansible-<deployment_name>-<deploy_id_lowercase>",
"app_dir": "/home/opc/siebel/microservices/siebel-ai-connectors/<deploy_id>",
"byo_ns": false,
"deploy_id": "<deploy_id>",
"deploy_status": "<creation-in-progress|completed|failed|Rerun-In-Progress>",
"infra_id": "<infra_id>",
```

```
 "name": "<mcp_deployment_namespace>",
 "namespace": "<mcp_deployment_namespace>",
 "product_name": "siebel-ai-connectors",
 "security_context": {
 "fs_group": <fs_group_id>,
 "run_as_group": <gid>,
 "run_as_user": <uid>
 },
 "self_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/siebelAIConnectors/<deploy_id>",
 "siebel_mcp": {
 "git": {
 "git_id": "<git_id>",
 "git_config_folder": "siebel-mcp",
 "git_repo_branch": "main",
 "git_protocol_type": "https",
 "git_url": "https://<git_host>/<group>/<repo>",
 "git_user": "<git_user>"
 },
 "java_truststore": {
 "cacerts_path": "/home/opc/siebel/cacerts",
 "password": "<masked_or_configured_truststore_password>"
 }
 },
 "stages": [
 {
 "name": "Pre Deploy",
 "stage_name": "pre_deploy",
 "status": "<passed|failed|in-progress>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/siebelAIConnectors/<deploy_id>/logs/
pre_deploy",
 "log_location": "/home/opc/siebel/microservices/siebel-ai-connectors/<deploy_id>/logs/pre_deploy.log"
 },
 {
 "name": "Prepare GitOps",
 "stage_name": "prepare_gitops",
 "status": "<passed|failed|in-progress>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/siebelAIConnectors/<deploy_id>/logs/
prepare_gitops",
 "log_location": "/home/opc/siebel/microservices/siebel-ai-connectors/<deploy_id>/logs/prepare_gitops.log"
 },
 {
 "name": "Verify Deployment",
 "stage_name": "verify_deployment",
 "status": "<passed|failed|in-progress>",
 "log_api_link": "https://<scm_host>:<scm_port>/scm/api/v1.0/siebelAIConnectors/<deploy_id>/logs/
verify_deployment",
 "log_location": "/home/opc/siebel/microservices/siebel-ai-connectors/<deploy_id>/logs/
verify_deployment.log"
 },
 {
 "name": "Publish Urls",
 "stage_name": "publish_urls",
 "status": "<passed|failed|in-progress>"
 }
 ]
 },
 "message": "success",
 "status": "success"
}
```


7. Delete Siebel AI Connectors Deployment

Use this API to delete an existing Siebel AI Connectors deployment.

```
API
DELETE /scm/api/v1.0/siebelAIConnectors/<deploy_id> HTTP/1.1
```

```
Host: https://<scm_host>:<scm_port>
Authorization: Basic <base64_admin_credentials>

Example Response
{
 "data": {
 "archive_dir": "/home/opc/siebel/microservices/siebel_mcp/archive/<deploy_id>",
 "deleted": true,
 "deploy_id": "<deploy_id>"
 },
 "message": "Deletion of deploy_id <deploy_id> of siebel_mcp successfull",
 "status": "success"
}
```

8. Perform Incremental Updates

Perform an incremental update after changes are made to the Siebel AI Connectors configuration files in the customer Git repository.

Configuration files may include either of the following files:

- application.yaml
- MCP_CONFIG.json
- OI_CONFIG.json
- FINAL_SPECIFICATION.json
- logging.properties

Steps:

1. Update the required configuration files in the customer Git repository.
2. Commit and push the customer Git changes.
3. Go to the infrastructure Helm chart repository.

   ```
   cd /home/opc/siebel/infrastructures/<infra_id>/<helmchart_repo>/
   git pull
   ```

4. Change to the image builder chart directory for the deployment.

   ```
   cd <deploy_name>-<deploy_id>/siebel-mcp
   ```

5. Edit `Chart.yaml`.

   ```
   vi Chart.yaml
   ```

6. Increment the chart version.

   ```
   version: 0.1.2
   ```

7. Commit and push the Helm chart change.

   ```
   git add Chart.yaml
   git commit -m "<incremental_update_message>"
   git push
   ```

Expected Result

After the Helm chart version is updated and pushed:

1. Flux reconciliation detects the Helm chart version change.
2. The MCP image builder job restarts.
3. A new Siebel AI Connectors application image is built using the latest committed customer Git content.

4. The updated MCP application starts with the new image.

Important Notes

1. Do not commit credentials, access tokens, registry passwords, truststore passwords, private keys, or other secrets into Git.
2. Ensure that all customer Git changes are committed and pushed before incrementing the Helm chart version.
3. Increment only the chart version required to trigger the image builder update.

Placeholder Reference

Placeholder	Description
<scm_host>	Siebel Cloud Manager host name or IP address.
<scm_port>	Siebel Cloud Manager port.
<infra_id>	Infrastructure ID returned by the infrastructure creation API.
<deploy_id>	Deployment ID returned by the Siebel AI Connectors deployment API.
<git_id>	Git repository ID returned by the Git registration API.
<git_host>	Git server host.
<group>	Git group, project, or organization path.
<repo>	Git repository name.
<git_user>	Git user name.
<git_access_token>	Git access token.
<infra_namespace>	Kubernetes namespace used for infrastructure resources.
<mcp_deployment_namespace>	Namespace or deployment name used for Siebel AI Connectors.
<registry_host>	Container registry host.
<registry_user>	Container registry user.
<registry_password>	Container registry password or authentication token.
<registry_prefix>	Registry namespace or prefix.
<uid>	Runtime user ID.
<gid>	Runtime group ID.
<fs_group_id>	File system group ID.
<truststore_password>	Java truststore password.
<kubeconfig_file>	Kubernetes kubeconfig file available on the SCM host.

Validating Runtime Behavior

Runtime behavior includes startup, configuration loading, MCP Server creation, tool invocation, and validation.

Runtime Startup

When the runtime starts, it performs the following actions:

1. Reads `application.yaml`.
2. Resolves the configured paths for `MCP_CONFIG.json`, `FINAL_SPECIFICATION.json`, `logging.properties`, and `OI_CONFIG.json`.
3. Loads logging configuration.
4. Loads Open Integration server configuration from `OI_CONFIG.json`.
5. Parses `FINAL_SPECIFICATION.json`.
6. Builds MCP tool definitions from the selected Open Integration REST API operations.
7. Loads MCP Server definitions from `MCP_CONFIG.json`.
8. Maps tools to MCP Servers.
9. Starts MCP Server endpoints.

Expected startup indicators include:

```
Successfully parsed OpenAPI spec file:<Path>/FINAL_SPECIFICATION.json
Loaded Open Integration servers: [<server-name-1>, <server-name-2>]; default server: <server-name>
Processing Tool : [<tool-name>] , Method : [<method>] , Path: [<path>]
Using Open Integration server [<server-name>] for tool [<tool-name>]
Adapted [<count>] tools from OpenAPI spec
MCP Servers started with base url: <host>:<port>/siebelAIConnectors/mcp
```

If a server URL in `FINAL_SPECIFICATION.json` differs from the server URL configured in `OI_CONFIG.json`, the runtime logs a warning and uses the configured Open Integration server URL.

Startup Validation

After startup, validate that the runtime loaded all required artifacts successfully.

Validation Check	Expected Result
`application.yaml` is loaded.	Runtime starts with configured host, port, security settings, and file paths.
`logging.properties` is loaded.	Logs use the configured handler, format, and logging levels.
`OI_CONFIG.json` is loaded.	Runtime logs the configured Open Integration servers and default server.
`FINAL_SPECIFICATION.json` is parsed.	Runtime logs successful parsing of the tool specification file.
MCP tools are generated.	Runtime logs the tools processed from the selected operations.
`MCP_CONFIG.json` is parsed.	Runtime creates MCP Servers with configured endpoint paths.
MCP Server endpoints are started.	Runtime logs the MCP base URL.

Tool Discovery and Invocation Validation

After startup validation, validate that MCP tools can be discovered and invoked.

Validation Check	Expected Result
MCP client connects to the MCP Server endpoint.	Connection succeeds.
Configured tools are available for discovery.	Tools listed in `MCP_CONFIG.json` are visible through the MCP Server.

Validation Check	Expected Result
Sample MCP tool is invoked.	Runtime identifies the tool and resolves the Open Integration server.
Open Integration REST API is invoked.	Runtime sends the downstream request to the configured Open Integration server.
Response is returned to MCP client.	MCP client receives the response from the tool invocation.

For secured deployments, also validate:

Validation Check	Expected Result
Request without token is sent.	Request is rejected.
Request with invalid token is sent.	Request is rejected.
Request with valid token and required scope is sent.	Request is accepted.
Token contains expected issuer, audience, and scope.	JWT validation succeeds.
Open Integration authentication is aligned.	Downstream Open Integration request succeeds.

Updating Existing Configurations

The runtime does not dynamically reload configuration artifacts. Restart the runtime after updating any of the following files:

- **`FINAL_SPECIFICATION.json`**
- **`MCP_CONFIG.json`**
- **`application.yaml`**
- **`OI_CONFIG.json`**
- **`logging.properties`**

It is critical to note that if the **`FINAL_SPECIFICATION.json`** is updated, then you must also review and if needed update **`MCP_CONFIG.json`** if tools were added, removed or renamed.

Troubleshooting Validation Issues

Use the following checks when runtime startup or tool invocation fails.

Issue	Check
Runtime does not start.	Verify Java version, runtime classpath, **`mcp-runtime-<version>.jar`**, **`libs/`**, and **`application.yaml`**.
Configuration file not found.	Verify **`mcp.server-config-file`**, **`mcp.final-spec-file`**, **`mcp.logging-properties-file`**, and **`mcp.open-int-config-file`**.
No tools are available.	Verify **`x-tools`**, operation-level **`x-tool-name`**, and **`MCP_CONFIG.json`** tool mappings.
MCP Server endpoint is missing.	Verify **`mcp-server-path`** in **`MCP_CONFIG.json`**.

Issue	Check
Tool mapping fails.	Verify every **mcp-server-tools** value exists in **FINAL_SPECIFICATION.json**.
Open Integration server is not resolved.	Verify operation-level **x-server-name**, **OI_CONFIG.json** server names, and **default-server**.
Request body validation fails.	Verify **requestBody**, **components.schemas**, **required**, and **additionalProperties**.
Parameter validation fails.	Verify parameter **in**, **name**, **required**, and **schema**.
JWT validation fails.	Verify **issuer**, **audience**, **required-scope**, **jwks-uri**, and token claims.
Open Integration invocation fails.	Verify Open Integration host, port, context, authentication, certificate trust, and network connectivity.

Using Siebel AI Connectors with Open Integration to Create MCP Servers

Siebel AI Connectors provide a configuration-driven framework for exposing selected Siebel Open Integration REST APIs as Model Context Protocol (MCP) tools. These tools can then be grouped into MCP Servers and consumed by AI agents and MCP-compatible clients.

Rather than requiring custom implementation to expose Open Integration REST APIs to AI agents, Siebel AI Connectors provide a configuration-driven approach for publishing selected APIs as MCP-compatible tools. These tools can then be organized into MCP Servers and consumed by AI agents and other MCP clients. For more details, see the Siebel AI Connectors topic in the Siebel Installation guide.