ORACLE

Siebel CRM - ODA Integration Approach

Technical Brief

October 2021

Public

Table of contents

1. Introduction

This technical brief is a practical guide for technical users who want to integrate Oracle Digital Assistant (ODA) with Siebel CRM. In this document, we will cover the architecture and the integration approach.

Before You Begin

This document is intended for developers with a basic understanding of Siebel CRM and ODA.

The integration requires a working ODA instance and Siebel CRM version of 21.x or above.

Importance of Digital Assistants for Siebel Customers

As business move towards more digital models, there is a gap that is to be filled for supporting modern channels where our customers can serve 'Digital Natives'. To bridge this gap, Digital Assistants are the way forward. Finely tuned Digital Assistants can provide a more personal and engaging experience as they help users perform rote tasks.

From our customer's perspective, they reduce their operating costs in supporting these journeys via agents or business development teams and drive efficiency and productivity.

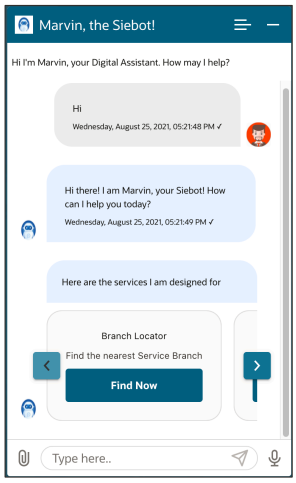
Oracle Digital Assistant is a platform that allows you to create and deploy digital assistants for your users. Digital assistants are virtual devices that help users accomplish tasks through natural language conversations, without having to seek out and wade through various apps and web sites.

Each digital assistant contains a collection of specialized skills. When a user engages with the digital assistant, the digital assistant evaluates the user input and routes the conversation to and from the appropriate skills.

You can populate your digital assistant with skills from the Skill Store and with skills you have designed yourself. You can make digital assistants available to users through a variety of channels, such as Facebook Messenger, Slack, and your own mobile apps.

Meet users where they are

As digital business models rise, brands need to transform Customer Service, driven by on-demand self-service and proactive engagement. They need to ease the administrative burden by automating repetitive tasks and enable single window for information access by employees. Digital Assistants can help in achieving all this.



Engage customers and employees with an AI-powered digital assistant that captivates users in a personalized manner, while delivering actionable insights to your sales, service, and marketing teams.

CUSTOMER VALUE

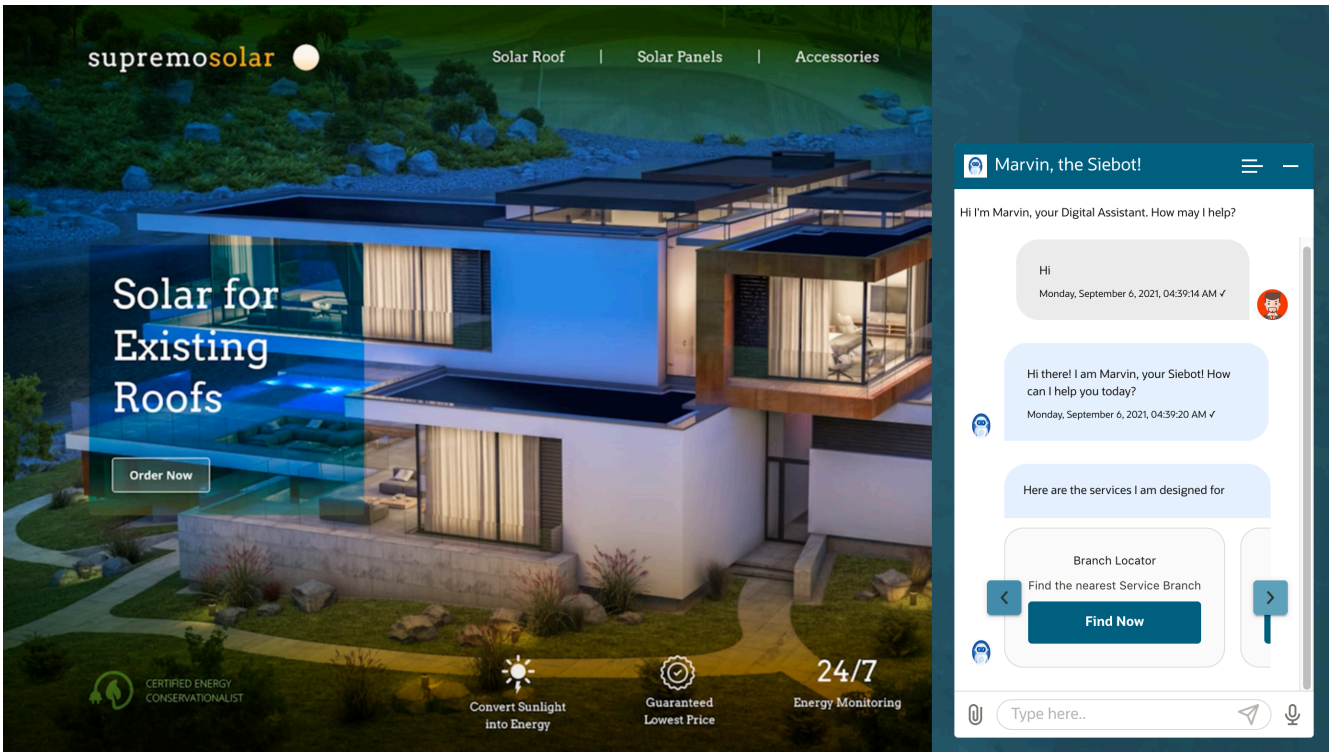
- Transform Service experience - reach customers / employees on new channels.
- Expand your reach to you employees and customers increasing operational efficiency and reducing cost.
- Get started with a library of ready to go skills; Build your own to meet your specific needs.

Basic Concepts

Before you dive into digital assistant and skill development, here are some concepts you'll want to get familiar with:

- **Intents** - Categories of actions or tasks users expect your skill to perform for them.
- **Entities** - Variables that identify key pieces of information from user input that enable the skill to fulfil a task.
- Both intents and entities are common NLP (Natural Language Processing) concepts. NLP is the science of extracting the intention of text and relevant information from text.
- **Components** - Provide your skill with various functions so that it can respond to users. These can be generic functions like outputting text, or they can return information from a backend and perform custom logic.
- **Dialog Flow** - The definition for the skill-user interaction. The dialog flow describes how your skill responds and behaves according to user input.
- **Channels** - Digital assistants and skills aren't apps that you download from an app marketplace, like iTunes. Instead, users access them through messaging platforms or through client messaging apps. Channels, which are platform-specific configurations, allow this access. A single digital assistant or skill can have several channels configured for it so that it can run on different services simultaneously.

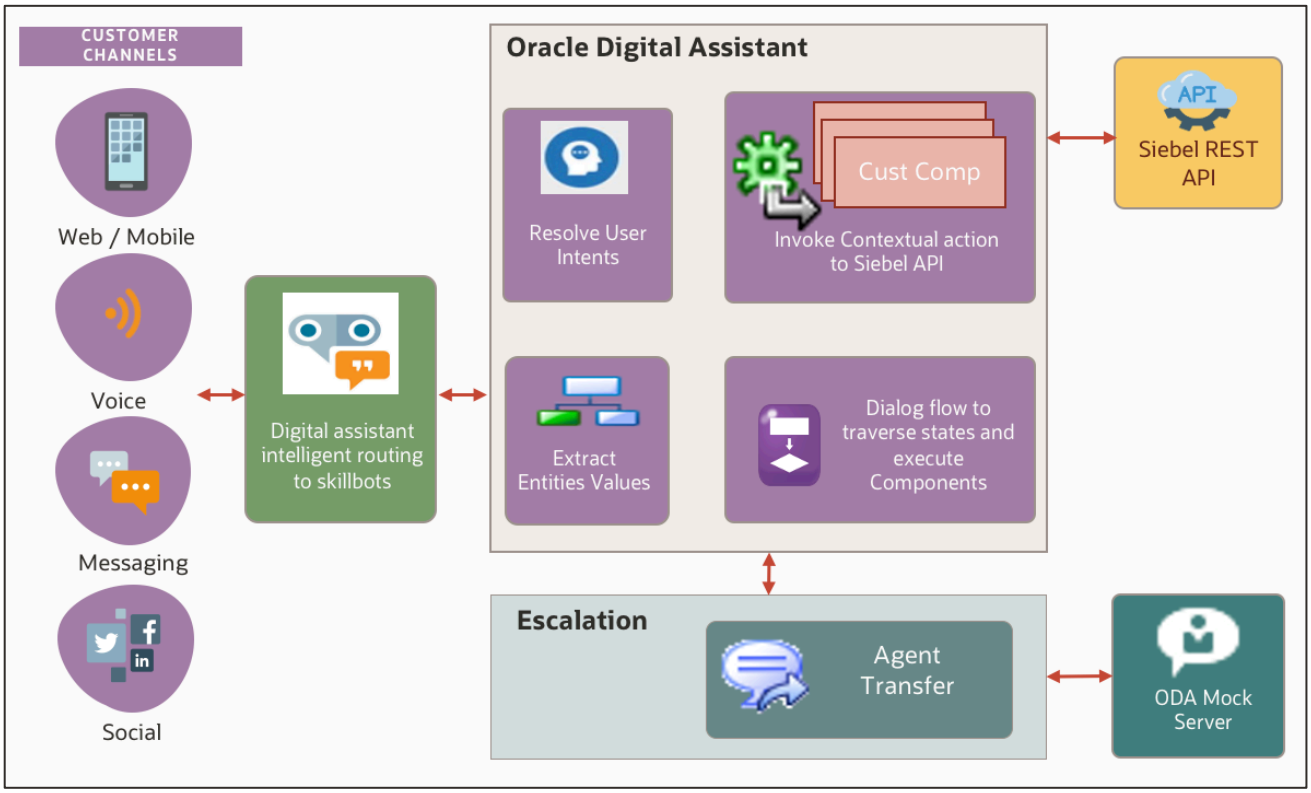
In this document, we will cover how Siebel users can integrate and embed Digital Assistant in Siebel's context.



2. Architecture

Technical Overview

A high-level overview of Siebel Integration with Oracle Digital Assistant is depicted in the below diagram. ODA is a cloud solution provided by Oracle and can be integrated with on-premise or on OCI Siebel Instances.



Key architectural components:

Chat Client: Chat client establishes connection with skills through the web channel. ODA provides a set of Javascript files named as WebSDK, which take care of construction of Chat Client within and HTML page or Siebel.

Skill: This refers to the actual bot deployed on ODA instance. This contains different Intents, Utterances and Dialogflow for the bot. The conversation between the user and bot is designed and defined in the skill. Refer to ODA documentation for more details.

Siebel Custom Components: Custom components are reusable units of code that can be called from skill's dialog flow. Custom component code is written using Javascript and NodeJS and is deployed in the ODA Node JS container. The custom components are local to a skill within ODA.

Two different skills in ODA with the same code deployed in them will run as two separate instances. The Siebel delivered skill comes with a packaged custom components code. The delivered custom component library and the package contains security logic and additional methods to facilitate ODA to Siebel communication.

Communication Flow

Digital Assistant routes the user requests coming from various channels to the Siebel skill bot. Here, user intents are resolved, and the interaction flow is determined. Custom components defined in skills, integrate the back-end Siebel system to perform specific tasks.

Also, in case of agent handover, the WebHooks are defined to manage the connection and establish a pipeline of messages between ODA and Agent's chat server.

Below is the step-by-step flow of information during the interaction between ODA and Siebel.

Step 1. Authenticate and Authorize the user

Step 2. Send user's message to ODA

Step 3. Route user's message to the relevant ODA skill

Step 4: Invoke Siebel REST APIs corresponding to the Intent

Step 5: Execute Siebel business logic and send the response back to ODA

Step 6: Digital Assistant processes the response and renders in the chat client

3. Developer Guide

Step 1: Identifying Intents and Conversation flows

Identifying Intents

Intents allow the skill to understand what the user wants it to do. An Intent categorizes typical user requests by tasks and actions that the skill performs. Intents are comprised of permutations of typical user requests and statements, which are also referred to as utterances. Intents are created by naming a compilation of utterances for a particular action. Because skill's cognition is derived from these Intents, each Intent should be created from a data set that is robust (one to two dozen utterances) and varied, so that the skill can interpret ambiguous user inputs. A rich set of utterances enables a skill to understand what the user wants.

Before starting actual development for Siebel skill or customizing an existing skill, it is important to recognize what changes are required. The first step is to decide the scope of the skill or change. Identify which business requirements are to be fulfilled. Once Identified, use that information to decide what are the Intents and entities required for your Skill.

Develop conversation Flow

Once Intents and Entities are identified, the next step in the design process is to develop the plan for conversational flow (Dialogflow). Conversational Design is the craft of imparting in a machine those human-like capabilities that enables the machine to interact with humans conversationally, on their terms – not on the machine's.

The easiest way to develop and describe the plan is to have a separate flowchart developed for each for the Intent resolution. Consider the different interactions and validations involved in the conversation. Identify, at which point in the interaction or the flow, Siebel Service calls are required for information or logic, and the possible Inputs, outputs and logical branching scenarios required after the service call.

Conversational design tactics

A few design best practices for conversational design:

- Limit open-ended questions
- Break responses into bite-sized chunks
- Offer more information in answers (e.g., hyperlinks)
- Confirm prior to segueing into transactional interactions
- If the chatbot goes off track, immediately escalate to an alternative

Required Rest Services

At the end of designing the conversation, the detailed flow charts for each of the Intent could be used by a developer to identify the different Application Services required and what should be the Input-Output Parameters and the Result States for the respective services.

Once the Services required have been identified, we need to ensure the corresponding Siebel REST Services are available. Please refer to *Siebel Rest API Guide* for details.

Step 2: Establish Skill Security

Refer to the chapter 'Skill Security' for detailed Security mechanisms available for Siebel CRM – ODA integration.

Step 3: Define Dialog Flow

The dialog flow definition is the model for the conversation itself, one that lets you choreograph the interaction between a skill and its users. Using the Flow Editor, you define the framework of the user-skill exchange in Siebel bot, Digital Assistant's own implementation of YAML. This is a simple markup language, one that lets you describe a dialog, both in terms of what your skill says and what it does.

Dialog Flow Structure

The skill definition is divided into two main parts – Context and States.

Context defines the variables that are available across the session.

States maintain the definition of the flow itself.

Define each part of the dialog and its related operations as a sequence of transitory states, which manage the logic within the dialog flow. To cue the action, each state node within the skillbot definition, names a component that provides the functionality needed at that point in the dialog. States are essentially built around the components. They contain component-specific properties and define the transitions to other states that get triggered after the component executes.

Components

Components give the Skill its actual functionality. The state nodes in the dialog flow definition are built around them. These reusable units of work perform all types of tasks and functions - from the basic management of the dialog flow to case-specific actions.

Every state in the dialog flow names a component that performs an action, such as accepting user input, verifying that input, or responding with text. Each component has a specified set of properties that can be used to pass and receive values as well as control the component's behavior.

A state definition might include the transitions that are specific to the component or the standard next, error, actions, or return transitions. Transitions describe how the dialog forks when variable values are either set or not set.

There are two types of components that can be used in a dialog flow – built-in components and custom components. When the Dialog Engine enters a state in the dialog flow, it assesses the component. When it encounters one of the built-in components, it executes one of the generic tasks, such as display a message or prompt the user to enter text. When the Dialog Engine discovers a custom component, however, it calls the component's service, which hosts one or more custom components.

Built-in Components

ODA provides a set of components that support a range of generic actions, which can be used in any skill: security, parsing the user input, routing the dialog flow based on that user input, and outputting the skill's responses in various ways.

To add a state for a built-in component to the dialog flow, click + Components on the Flows tab, select the component type, and then select the component. The component's template is displayed, which lists the component's properties with descriptions.

When you validate your dialog flow, Oracle Digital Assistant verifies the component's properties. For example, it will report if you forgot to include a required property.

Custom Components

Most skills need to integrate with data from remote systems or do some backend processing. For example, service request skill needs to get the status of your service requests from Siebel server and display the same in the conversation. Custom components enable you to integrate with backends as well as perform tasks that aren't covered by the built-in components.

Here's an example of a custom component state in the dialog flow.

```
203
204  #Call the custom component
205  createSR:
206    component: "createSR"
207    properties:
208      emailAddress: ${email}
209      product: ${category}
210      srType: ${srtype}
211      description: ${description}
212    transitions:
213      actions:
214        createSRSuccess: "createSRSuccess"
215        createSRFail: "createSRFail"
216
```

createSR is the name of the custom component which takes the variables email address, product category, SR type and description as the input. Based the execution status of the custom component, it will transit next to a success or failure state.

Understanding Custom Components Structure

The ODA architecture expects the custom components to be JS modules (usually each component is a different module). Every custom component could be said to be an implementation of the following interface:

```
      // interface for a custom component implementation
{
  metadata(): {name: string, properties?: {[name:string]: string},
supportedActions?: string[]};
  invoke(conversation: Conversation, done: () => {}): void;
}
```

The following is the code sample for a greeting Custom component with input property “name”. The custom component uses the conversation object to send a reply to the chat client with a customised greeting that contains a name in it.

```
module.exports = {
   metadata: () => ({
     name: 'greeting',
     properties: {
       "name": {
         "type": "string",
         "required": true
     }
  }
}),
invoke: (conversation, done) => {
  // Get property values
  const { name: _name = '' } = conversation.properties();
  conversation.reply('Hello ${_name}');
  conversation.transition();
  done();
  }
```

Note: Please refer to the ODA documentation for understanding the custom components further in detail.

Step 4: Developing Custom Components for Siebel Integration

Custom Components

Custom components are re-usable units of work that are defined within each state node of the Dialog flow. However, unlike the built-in components, custom components perform actions that are specific to the bot. They execute functions that the system components cannot perform. You can "package" related custom components together, into a component service.

Common activities performed by custom components include:

- Modify/Change the variable value in the incoming payload sent from the Dialog flow.
- Implement custom logic/rules.
- Invoke the target Siebel REST API end point for taking action or getting the necessary data from Siebel.

Install Oracle Bots Node.js SDK

The Oracle Bots Node.js SDK is a Node module for building and deploying custom component services for Oracle Digital Assistant.

1. To install the Oracle Bots Node.js SDK for global access, enter the following command in a terminal window:
   npm install –g @oracle/bots-node-sdk
2. To verify the success of the installation, enter this command and check the version: bots-node-sdk –v

Create a Custom Component Package

You can use the bots-node-sdk init command to create a package structure with the necessary files, including a custom component and the javascript file.

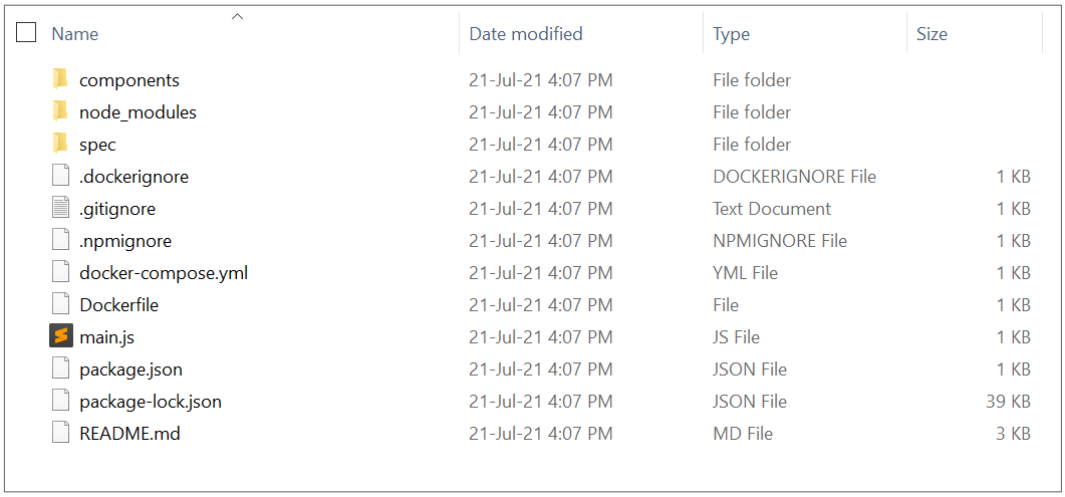
For example: we can use this command “bots-node-sdk init converterccs –name converterccs –component-name CurrencyConverter”, to create a currency converter package.

The command creates a subfolder named coverterccs, which contains main.js and package.json files, along with the installed dev dependencies in node_modules. It also creates a components folder that contains CurrencyConverter.js file.

For our Service Request custom component, we can use the below command to generate the folder structure:

bots-node-sdk init SiebelSR –name SiebelSR –component-name ServiceRequest

A screenshot of the folder created by this command:



Name	Date modified	Type	Size
components	21-Jul-21 4:07 PM	File folder	
node_modules	21-Jul-21 4:07 PM	File folder	
spec	21-Jul-21 4:07 PM	File folder	
.dockerignore	21-Jul-21 4:07 PM	DOCKERIGNORE File	1 KB
.gitignore	21-Jul-21 4:07 PM	Text Document	1 KB
.npmignore	21-Jul-21 4:07 PM	NPMIGNORE File	1 KB
docker-compose.yml	21-Jul-21 4:07 PM	YML File	1 KB
Dockerfile	21-Jul-21 4:07 PM	File	1 KB
main.js	21-Jul-21 4:07 PM	JS File	1 KB
package.json	21-Jul-21 4:07 PM	JSON File	1 KB
package-lock.json	21-Jul-21 4:07 PM	JSON File	39 KB
README.md	21-Jul-21 4:07 PM	MD File	3 KB

Inside the *components* folder, ServiceRequest.js file will be generated. The logic and code for the custom components needs to be written in this file.

```
'use strict';

// You can use your favorite http client package to make REST calls, however, the node fetch API is pre-
// Documentation can be found at https://www.npmjs.com/package/node-fetch
// Un-comment the next line if you want to make REST calls using node-fetch.
// const fetch = require("node-fetch");

module.exports = {
  metadata: () => ({
    name: 'ServiceRequest',
    properties: {
      human: { required: true, type: 'string' },
    },
    supportedActions: ['weekday', 'weekend']
  }),
  invoke: (context, done) => {
    // Retrieve the value of the 'human' component property.
    const { human } = context.properties();
    // Determine the current date
    const now = new Date();
    const dayOfWeek = now.toLocaleDateString('en-US', { weekday: 'long' });
    const isWeekend = [0, 6].indexOf(now.getDay()) > -1;
    // Send two messages, and transition based on the day of the week
    context.reply(`Greetings ${human}`)
      .reply(`Today is ${now.toLocaleDateString()}, a ${dayOfWeek}`)
      .transition(isWeekend ? 'weekend' : 'weekday');
    done();
  }
};

```

The integration with a remote backend system by the help of a Rest API can be done with the help of this javascript file. For example, in case of Siebel Service Request, we will use the following rest API: https://<server_name>:32701/siebel/v1.0/data/Service Request/Service Request/?[search spec]

Service Request ID from the ODA user is passed in the search spec of the above URI. The Service Request summary returned by Siebel can be output in various forms like, text or a “Card Format” (for specific details, please refer to ODA documentation).

Package and Deploy the Custom Component Service

You can deploy custom components to either a skill’s embedded container, Oracle functions, a remote node server, or Oracle Mobile Hub.

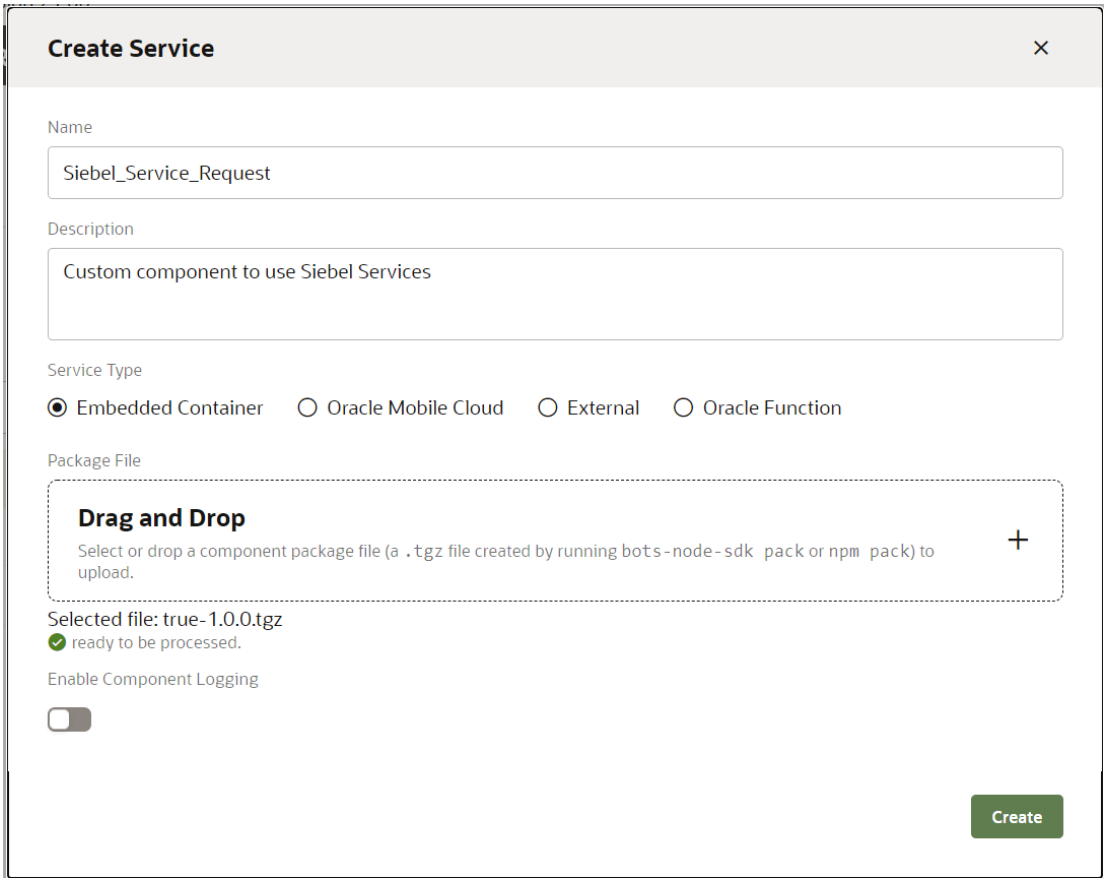
1) Save your work in your Javascript IDE
2) Open a terminal window and navigate to your components folder, in this case the SiebelSR folder.
3) In the SiebelSR folder, enter this command: “bots-node-sdk pack”

The command then packages the Node Project and its dependencies into a deployable tarball, true-1.0.0.tgz as shown below:

```
C:\Users\shubhakk\SiebelSR>bots-node-sdk pack
-----------------------------------------------------------------
Preparing artifact from: SiebelSR...
-----------------------------------------------------------------

> true@1.0.0 prepack
> npm run bots-node-sdk -- pack --dry-run


> true@1.0.0 bots-node-sdk
> bots-node-sdk "pack" "--dry-run"

-----------------------------------------------------------------
Component Package 'SiebelSR' is valid!
-----------------------------------------------------------------
npm notice
npm notice package: true@1.0.0
npm notice === Tarball Contents ===
npm notice 30B   .dockerignore
npm notice 86B   Dockerfile
npm notice 2.6kB README.md
npm notice 1.1kB components/ServiceRequest.js
npm notice 173B  docker-compose.yml
npm notice 60B   main.js
npm notice 568B  package.json
npm notice === Tarball Details ===
npm notice name:          true
npm notice version:       1.0.0
npm notice filename:      true-1.0.0.tgz
npm notice package size:  2.4 kB
npm notice unpacked size: 4.7 kB
npm notice shasum:        6077fba44d0eb753da293668b020a2a045a76709
npm notice integrity:     sha512-aUHRr4oYntu24[...]xQl7nqda8tkHQ==
npm notice total files:   7
npm notice
true-1.0.0.tgz
-----------------------------------------------------------------
Component package 'true-1.0.0.tgz' created successfully!
-----------------------------------------------------------------
```

Deploying the Custom Component Service to your Skill

1) After creating a skill, click Components to open the component service page.
2) Click +Add Service to open the Create Service Dialog.
3) In the *Name* field, enter the custom component name, in this case we will use *Siebel Service Request.*
4) Enter the appropriate description.
5) Ensure that *Embedded Container* is selected.
6) Locate the true-1.0.0.tgz file in the SiebelSR folder on your file system
7) Drag and drop the true-1.0.0.tgz file into the Create Service dialog's *Package File* field.
8) Switch *Enable Component Logging* to On.
9) Click *Create.*



Step 5: Testing and Publishing a skill

The *Conversation Tester* lets you simulate conversations with your skill to test the dialog flow, intent resolution, entity matching, and Q&A responses. You can also use it to find out how conversations would render in different channels.

To start the Conversation Tester:

1. Open the skill that you want to test.
2. At the top of the page near the Validate and Train buttons, click play button.

3. To preview how a skill renders on a given channel, select the channel type from the Channel dropdown. The Conversation Tester simulates how the skill behaves within the limitations of a given channel. By default, the Conversation Tester simulates the Webhook, which renders the UI per the Oracle Web SDK.
4. In the text field located at the bottom of the Tester, you can either choose the language for the entire conversation from among the supported languages configured for the skill or click Detect Language option for the skill to automatically detect the language of your text or spoken message.
5. For voice, you need to click Speak for each voice command. Click Attach to test a file, audio, video, or image attachment response. For example, you can use Attach to test an attachment response rendered by the System.CommonResponse component.

Typically, you'd use the Conversation Tester after you've created intents and defined a dialog flow. It's where you actually chat with your skill or digital assistant to see how it functions as a whole, not where you build Q&A or intents.

Track Conversations

In the Conversation tab, the Tester tracks the current response in terms of the current state in the dialog flow. Depending on where you are in the dialog flow, the window shows you the postback actions or any context and system variables that have been set by a previous postback action. It also shows you any URL, call, or global actions.

In the Intent/Q&A tab, you can see the resolved intent that triggered the current path in the conversation. When the user input gets resolved to Q&A, the Routing window shows the ranking for the returned answers. If the skill uses answer intents for FAQs, then only the resolved answer intent is displayed.

Finally, the JSON window shows you the complete details for the conversation, including the entities that match the user input and values returned from the backend. You can search this JSON object or download it.

Publishing a Skill

When you've completed building a version, you can lock it down by publishing it. The only modification that you can make for a published skill is to change custom parameter values in the Configuration tab. If you want to make further modifications, you must create another version and work on that one.

To publish a version:

1. If the skill has intents or Q&A, make sure it has been trained. You must train it before you can publish it.
2. From the Skill Catalog, locate the version that you want to publish.
3. Click the Options icon and select Publish.

The skill version in the Skill Catalog now has a lock icon to show that it's published.

Step 6: Reusing Skills

You can extend any skill that you have pulled from the ODA skill store to customize it for cases specific to your business. When a new version appears in the Skill Store, you can transfer your customizations to the new version by rebasing.

What is Extension & What's it for?

When you install bots from the skill store, they may not satisfy all your requirements, or you may want to modify them to align with your business processes. You can't modify an installed bot directly, but you can create an extension of it and then modify that extension.

When you create an extension, you are creating a new bot that has a tight relationship to the original (base) bot. Through this relationship, you can later take advantage of updates to the base bot without having to manually reapply your customizations. You do this by using the Rebase feature. When a new version of the base bot becomes available in the Skill Store, you can install that version into your instance and then rebase your extended bot to the updated base version.

Cloning vs Extending

Though cloning and extending are similar on a surface level, they have key differences and purposes:

When you create a clone of a bot:

- You create a totally independent copy of the bot.
- You can make unlimited changes to the clone.
- The clone loses all association with the original bot (the tracking IDs for the cloned bot do not match those of the original), so you can't later rebase to an updated version of the original bot.

Use cloning when you want to use an existing bot as a starting point for your development.

When you extend a bot:

- You can make a wide range of additions and changes to the extended bot but you cannot delete anything that was defined in the base bot.
- You can later rebase, which means applying updates from the base bot into your extended bot. Rebasing is possible for extended bots because the internal tracking IDs that are generated for the extended bots match those of the base bots.

Use extension when you want to customize a bot and then later be able to incorporate any improvements or new features from the base bot into your customized version. You can only extend skills that you have pulled from the Skill Store.

Extend a skill

Here's what you need to know about extending skills that you have pulled from the Skill Store.

1. Click ☰ to open the side menu, and select Development > Skills.
2. In the tile for the skill that you want to extend, click the Options icon (☰), and select Extend.

The skill to be extended should be pulled from the Skill Store.

What You Can Add and Customize in an Extended Skill

Intents: You can add utterances, change existing utterances, and add new intents. You can't delete utterances or intents, but you can disable intents.

Entities: You can add entity values, add synonyms to entity values, and add new entities. In addition, you can edit these fields:

- Enumeration Range Size
- Error Message
- Multiple Values
- Fuzzy Match

However, you can't delete entities or delete or change entity values.

Dialog Flow: You can make changes throughout the dialog flow. There are no specific limitations. However, no deltas are tracked by the system. When you rebase the skill extension, you are presented with a diff tool to compare your dialog flow side-by-side with that of the new base skill. It's then up to you determine what to keep from your skill and what to bring forward from the new base skill.

Resource Bundles: You can add new message keys in any of the supported languages and modify any of the existing messages.

Custom Component Service: You can replace the package file and add components to the service. You can't remove existing components. You can change the implementation of custom components in your extended skill. However, if the custom component is later updated in the base skill, those updates will not be merged with any changes you have made to the component in the extended skill when you rebase your skill. In this case, you would need to manually merge the custom component changes from the updated base skill into your extended skill.

Settings: You can adjust most of the settings for the skill, including:

- General properties, like skill description.
- Training model.
- Whether insights and conversation logging are enabled.
- Values of system parameters, such as confidence threshold and standard prompts.
- Custom parameters. (You can create new custom parameters and modify values of existing ones.

Disable Intents

When you extend a skill, you can't remove intents, but you can disable them.

When you disable an intent, you exclude it from the training model. Any user input that would otherwise match well with a disabled intent's training data will instead resolve to a different intent (likely unresolved Intent).

If you later rebase the skill, any intents that you have disabled will remain disabled. If you re-enable an intent after rebasing, you will pick up any changes that were made to that intent in the base skill.

4. Using Predefined Skills

Service Request Skill

Introduction

The sample Service Request skill is designed to enable customers to raise a new service request, to get the real time status on their service requests, to locate a service center within area covered by a pin code and view product demo videos. With the help of this chatbot, customer can get address and contact details of the registered service center in Siebel database, they can get instant status on the given SR number or they can get status on all their recent SRs raised in one go. Based on the status, customers can take appropriate actions like escalate an SR, talk to an agent in case they are not satisfied. The skill consists of multiple intents where each intent is responsible to fulfill certain user queries.

Intents

Intents allow the skill to understand what the user wants it to do. The service request skill consists of the following intents:

S. No.	Intent Name	Function	Example Utterances	Entities
1	WelcomeMessage	Greets the customer and prompts user with the probable suggestions to get started	Hello. Hi! Can you help me? Good Morning!	None
2	RaiseSR	Verifies the user and creates a new service request based on user inputs	I want to raise an SR I want to create to Service Request Help me in creating an SR	EmailAddress
3	StatusEnquiry	Takes SR number as the user input and displays the status	I raised a Service Request I want to enquire about my issue Service Request Status? Complaint Status? What is the status of my SR 296-4563452?	SR_Number
4	ShowProdDemo	Shows a list of demo products and displays the video based on user selection	Can I see the video for the product? Can you show me a demo? Are there any demos available? I want to see demo of your product	None

S. No.	Intent Name	Function	Example Utterances	Entities
5	BranchLoc	Takes zip code as the input and provides service centers in that area	Can you give me a service center address? Find me Service Centre near 92704	ZipCode
4	Unresolved	Triggers when skill is unable to find any corresponding intent.	It doesn't help me This doesn't make sense	None

Entities

S.No.	Entity Name	Type	Value
1	SR_Number	RegularExpression	Defines the value of a service request number
2	SRStatus	ValueList	Approved, Closed, In-Progress, Open, Rejected
3	EmailAddress	Regular Expression	Defines the value of user email address
4	SRType	Value List	Grid Connection, Panel, Panel Monitor, Peripherals
5	Product	Value List	Supremo Charge Controller, Supremo Microgrid, Supremo Roof Panel, Supremo Tubular Battery
6	ZipCode	Regular Expression	Defines the value zip code of an address

Custom Components

SRUserInquiry

This component invokes Siebel REST service and retrieves the service request status based on the user parameters. It takes user and idcsCCToken for authorization as the inputs and consists of the following actions:

1. MyRequestsList - Returns the list of recent SRs logged by the user (currently set to a max value of 5).
2. myRequestsListTopFive – In case there are more than five records, the parameter returns the list of top five SRs.
3. TooManySRs - Asks user about the SR number if the recent SR count is greater than the max value count.
4. SRSummary - Returns the summary of the SR.
5. Failure - Any failure in getting the SR status will result in transferring the control to the agent.

SRSummary

This component is responsible for returning SR status for a particular SR number. It takes input parameters as idcsCCToken for authorization and SR Number and returns the status of the same in case of a success and returns failure otherwise.

verifyAccount

This component verifies a given user based on the provided mobile/email. It returns the success if the user is verified otherwise returns a failure.

createSR

With the help of this component, a new service request is raised. It takes user input parameters like product category, SR type, description, idcsCCToken for authorization and creates a new service request for that user. It returns the new SR Number created on success and returns failure in case something goes wrong.

getDemoURL

This component returns the demo URL for the selected product by the user. It takes idcsCCToken for authorization and product id as input. The component will return failure in case the requested product video URL not found, or any error occurred.

getAddress

This component invokes a rest service query to Siebel CRM to get the service centers based on the user parameters. It takes zip code and idcsCCToken for authorization as input. It consists of the following actions:

fetchSuccess - Returns the list of service centers in a card layout.
fetchFailure - Triggered when no service center is found.

5. Agent Handover

This section provides a view on escalation of user chat from ODA skill to a live agent and back to the ODA skill.

There could be situations where the bot is unable to understand user intent or user may not be satisfied with the responses of the bot. In such scenarios, chat can be transferred from the bot to a live agent, who can converse with the user, perform actions, and can shift control back to the bot.

Customers can leverage a Chat Server for which, REST APIs are available for the integration.

Components involved:

Component	Description
Bot User	The end user chatting with the ODA bot.
Agent	Agent conversing with the user using Chat Server.
ODA	The ODA bot, that will be communicating with the User as well as the WIS to transfer user requests to live agent.
Webhook Implementation Server (WIS)	This server acts as an intermediary between the ODA and the Chat Server. It is responsible for processing all the incoming requests and sending them to live agent. The whole communication between the user and the agent happens through WIS.
WIS Cache	The WIS cache stores user session info that is necessary for the communication between user and the agent.
Chat Server	The live agent will be conversing with the user through Chat Server.

Control Flow for Chat Escalation

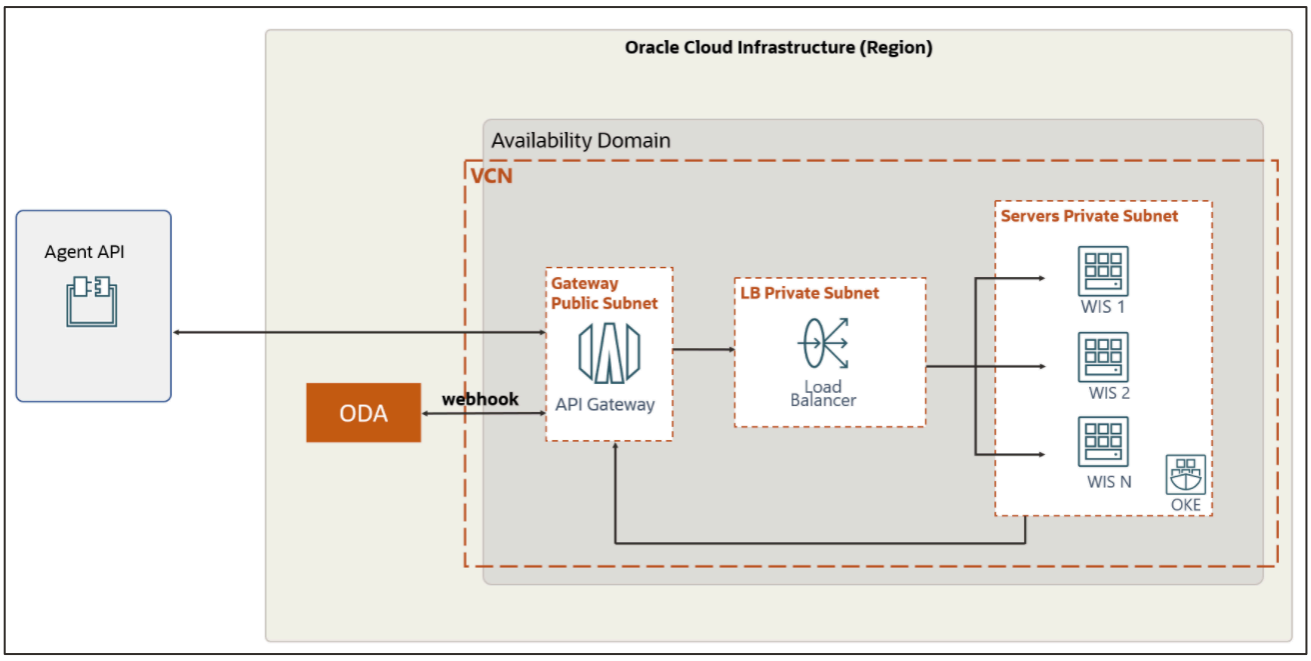
A sample interaction flow between ODA, WIS and Chat Server will follow the following steps:

1. User requests chat to be escalated to a live agent.
2. ODA instance initiates a POST request using the webhook channel with the necessary payload, requesting chat with a live agent. If the user is already in conversation with the agent, payload will contain the user text message.
3. WIS processes the request, interprets the incoming chat, and extracts user info from the payload.
4. Stores userId and webhook channel details in the WIS cache.
5. Transforms the payload to a Chat Server compatible payload.
6. Sends the POST request to the Chat Server.
7. Receives POST request from Chat Server, which contains information about the agent session including acceptance or denial of chat.
8. Retrieves user info from WIS cache using the userId.
9. Maps the userId with the sessionId in the WIS cache, that enables conversations to be directed to the right user and agent in ODA and Chat Server respectively.
10. Transforms the received payload to the ODA webhook channel compatible payload.
11. Sends the agent's response (transformed payload) to ODA instance.
12. Displays the message to user.

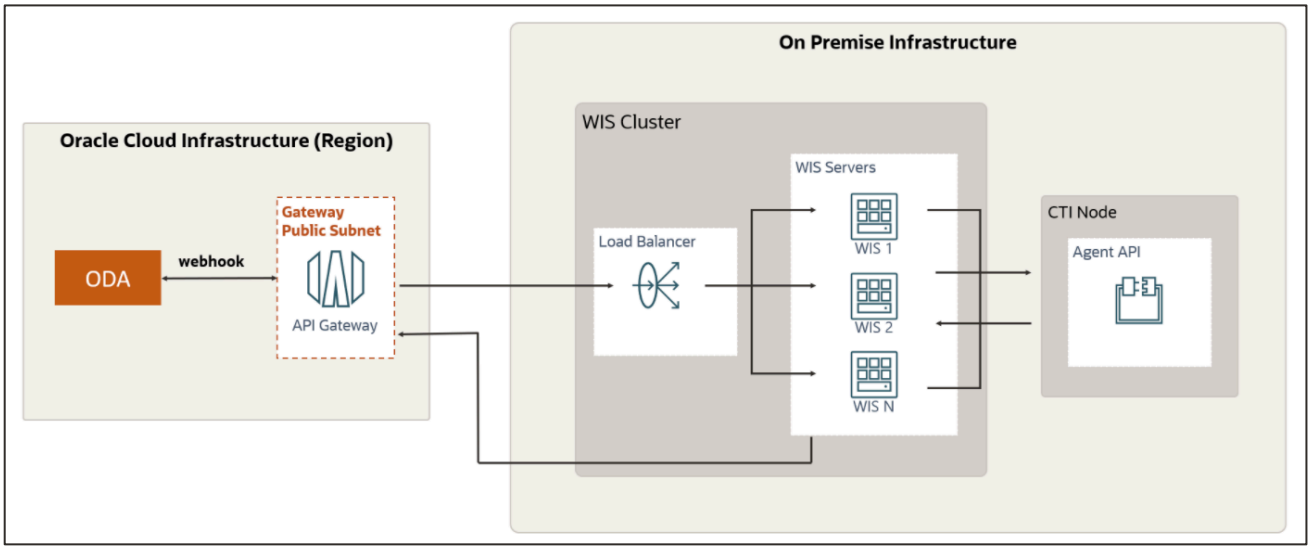
Webhook Deployment

The WIS can be deployed using two possible infrastructures:

1. Deployment on Oracle Cloud Infrastructure



2. On-Premise Deployment



Architecture Components:

Component	Description
ODA	ODA instance which contains the skill from which the transfer is initiated.
WIS Servers	Managed by the Oracle Kubernetes Engine (OKE). These lie in a private subnet, only accessible by resources within the VCN.
API Gateway	Used to enable communication with the Chat Server APIs.
Oracle Kubernetes Engine (OKE)	Responsible for autoscaling the WIS servers, depending on load requirements.
Load Balancer	Responsible for distributing the load between the pods managed by OKE.
Webhook channel	Webhook configured by ODA instance to communicate with the WIS to transfer messages.
Agent API	APIs for agent communication.

6. Skill Security

Chatbots created with ODA integrate with remote back-end systems through custom components that invoke REST services. For custom components to access protected REST endpoints, some sort of authorization must be passed in the request header.

OAuth 2.0 (Open Authorization) is the standard protocol for token-based authorization. It allows clients (such as chatbots) to access protected resources on behalf of a resource owner without passing the resource owner's credentials with the request. The two most commonly used authorization options in OAuth2 are:

- **Client Credential Flow** - Using the Client Credential Flow, clients like ODA obtain authorization to protected resource through a shared client Id and client secret. This authorization flow type can be handled using a custom component only.
- **Authorization Code Flow** - Authorization Code Flow requires a bot user to authorize a resource access. At first, client requests the authorization token for which it needs the user to login to the remote identity provider and to grant the requested access (defined as "scope"). With the authorization token, using a custom component, ODA then requests a second endpoint to exchange the authorization token. The access token then needs to be sent with each access request to the protected resource.

Security Components

- **System.OAuth2AccountLink**

Use this component to obtain an OAuth2 user access token (grant type Authorization Code) for resources that are secured by Oracle Identity Cloud Service (IDCS), Oracle Access Manager (OAM), Microsoft identity platform, or Google OAuth 2.0 authorization. This component completes all the steps for the 3-legged OAuth2 flow and returns the OAuth2 access token.

- **System.OAuthAccountLink**

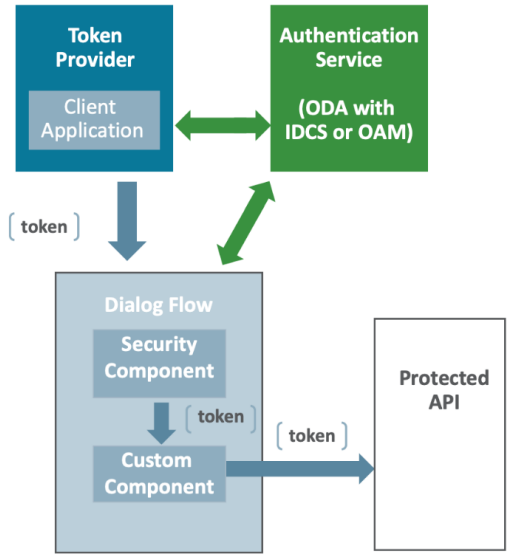
Use this component to obtain the authorization code for services that are secured by the authorization code grant flow, such as LinkedIn, Twitter, Google, or Microsoft. The skill's custom components can exchange the authorization code for an access token, which they then use to invoke the end service.

- **System.OAuth2Client**

Use this component to obtain an OAuth2 access token of grant type Client Credentials. That is, you use it to get an access token that's based on the client's credentials, and not the user's name and password. You can use this component to get a token that enables access to client resources that are protected by Oracle Identity Cloud Service or Oracle Access Manager (OAM).

ODA OAuth 2.0 Security Flow

When user initiates the conversation with ODA through channels, such as SMS, ODA routes the request to skill bot, if skill bot has the **System.OAuth2AccountLink** component configured. ODA will check whether user is already logged in with a valid access token. If not, it will send back OAuth login link for the user to login.



OAuth Components

Terms	Meaning	Example
Resource Owner	Person or application that owns the data that is to be shared.	Bot End Users • Employees • End Customers
Resource Server	Server hosting the resources	Siebel Server
Client Application	Application requesting access to the resources stored on the resource server.	Registered with IDCS or any third party IDP; ODA configured as Client with in IDP
Authorization Server	Server authorizing the client app to access the resources of the resource owner.	IDCS or any third party IDP

Siebel OAuth Enablement

Siebel supports all OAuth 2.0 authentication flows. Siebel Object Manager must be configured for SSO when OAuth is enabled. The required certificates from the OAuth server must be installed in the environment where the Siebel REST API is hosted.

Following table describes Siebel Application Interface profile configuration parameters:

Parameter	Description	Value
SingleSignOn		TRUE
Authentication Type	Specify the authentication type that the Siebel Application Interface nodes accept for REST inbound authentication. The options are: • Basic Authentication • Single Sign-On	OAuth

Parameter	Description	Value
	• OAuth	
Trust Token	Specify the trust token, which will be used as the password when Single Sign-On or OAuth is enabled. The specified value is passed as the Password parameter to a custom security adapter, if the value corresponds to the value of the Trust Token parameter defined for the custom security adapter.	same as the security adapter TrustToken
Authentication URL	Specify the URL to use for REST inbound authentication (OAuth). It is recommended that you specify the URL using the HTTPS format.	URL of the OAuth Service Provider end point for Access Token validation
Secure Channel	This option applies only for the OAuth authentication type as follows: • Select this check box only when you have already imported the Authentication URL's CA certificate into the Application Interface truststore. • Deselect this check box when the Authentication URL's CA certificate is not available in the Application Interface truststore. In this case, the Application Interface trusts all certificates while calling the Authentication URL over HTTPS.	

Siebel Settings

1. Enable DB authentication with SSO for EAI OM
    a. Connect to the server manager and set the following parameter values for the EAI: SecAdptMode=DBSSO and SecAdptName=DBSecAdpt.
        - change param SecAdptName=DBSecAdpt for component EAIObjMgr_enu server <server name>
        - change param SecAdptMode=DBSSO for component EAIObjMgr_enu server<server name>
    b. Add the database adapter advanced subsystem parameter values described in the following table for SSO support. These parameters are required in addition to other existing parameters
        - change param DBSecAdpt_SharedDBUsername=SADMIN for named subsystem DBSecAdpt
        - change param DBSecAdpt_SharedDBPassword=SADMIN for named subsystem DBSecAdpt
        - change param DBSecAdpt_SingleSignOn=True for named subsystem DBSecAdpt
        - change param DBSecAdpt_TrustToken=IDCSSIEBEL for named subsystem DBSecAdpt

2. On SMC, change the following settings and deploy
    a. AI Profile → General Section → Rest Inbound Authentication
    Anonymous UserName = guestcst
    Anonymous Password = guestcst
    AuthenticationType = OAuth
    Authentication URL = Append base64-encoded[client-id:client-secret-string] to IDC introspect URL
    ex https://<idcs_server>/oauth2/v1/introspect?YzFjNDViM2I4MjM2NGI0MWFlZGNlYjc0YjNiNTIzZWU6OTg3NDgyNGUtNjM5NC00YWRjLTg2NWEtMDAwZTliZGNmYTJm

Trust Token = IDCSSIEBEL
Session Timeout = 120
Secure Channel is not selected

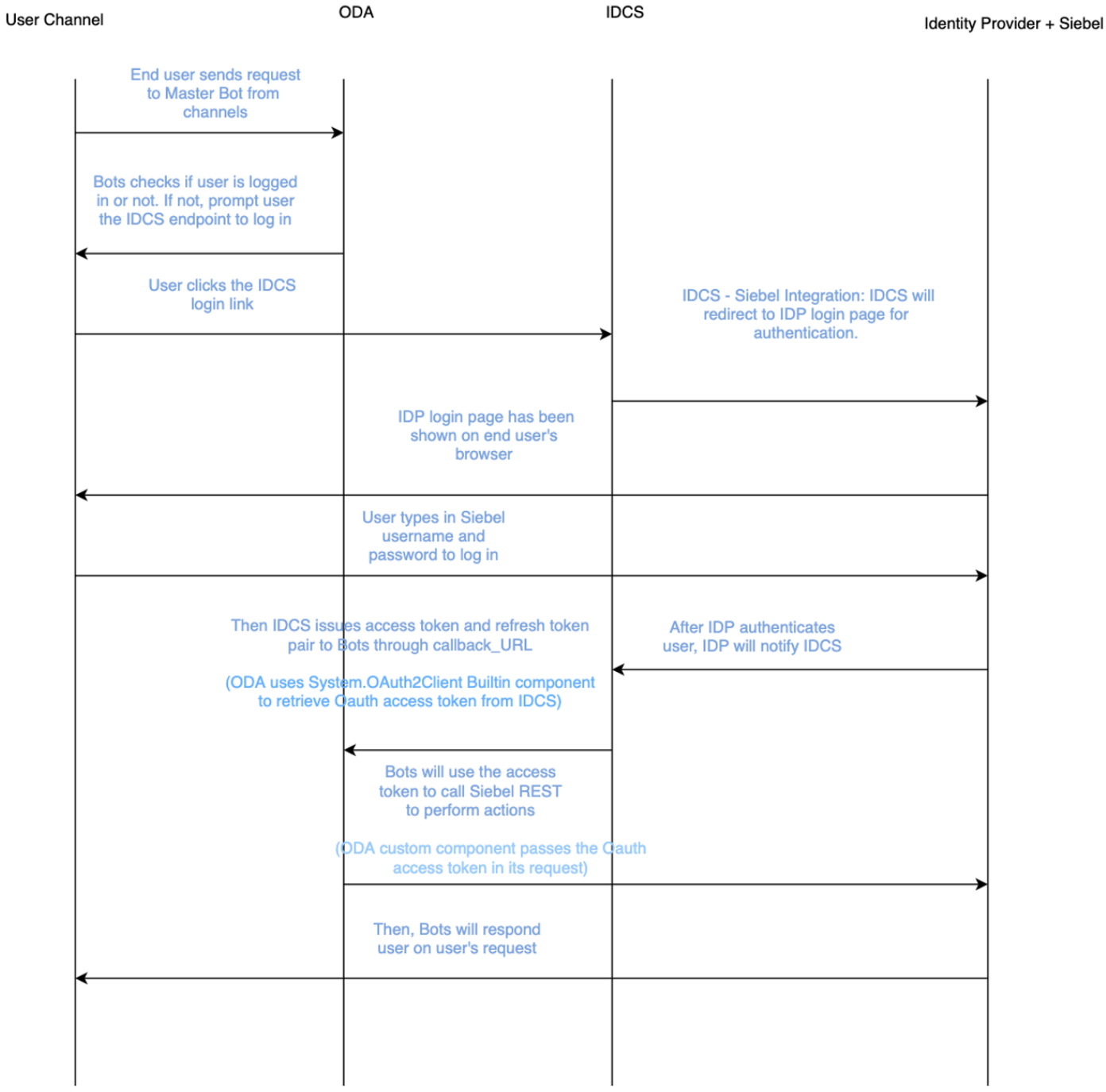
b. Under Applications → For EAI
Anonymous User Name: guestcst
Anonymous User Password: IDCSSSIEBEL
Keep the rest of the settings as default.

3. For Client credentials, the User Id matching with ClientID should be created in Siebel.

Refer ***Siebel Security Guide*** and ***Siebel Rest API guide*** for OAuth 2.0 authentication flows.

Authorization Flow

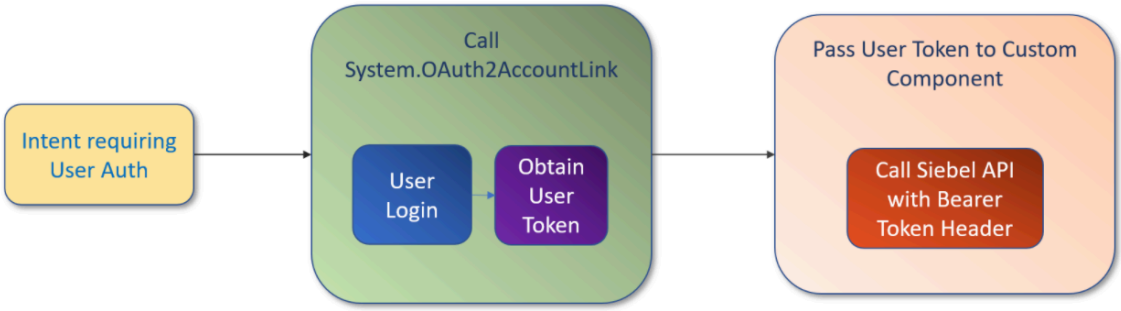
Authorization code flow



Siebot Authorization Types

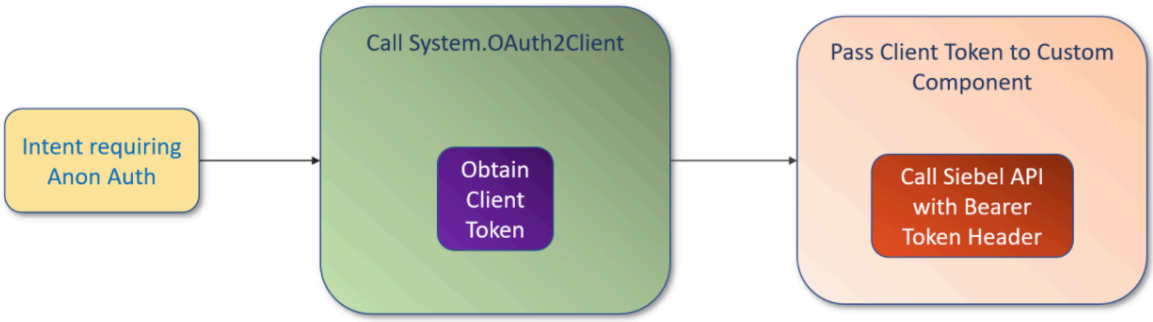
Access through User SSO

- Intent is configured with **System.OAuth2AccountLink**
- ODA directs user to configured OAuth browser login link for user to login
- Once Authentication is successful, user can continue the conversation
- The token obtained through authentication is sent to Siebel API for any resource requests



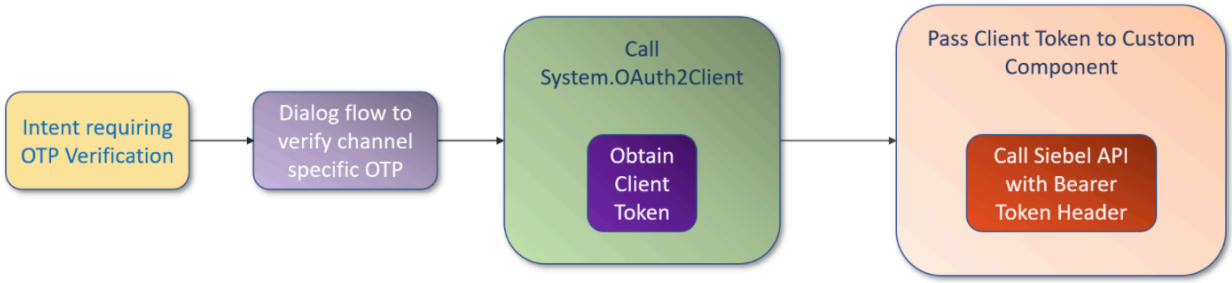
Anonymous Access with Client Token

- Intent is configured with **System.OAuth2Client**
- ODA obtains token for the Siebel client registered in IDCS without any inputs from user
- The token obtained is sent to Siebel API for any resource requests



Channel OTP Verification

- Intent is configured with System.OAuth2Client
- ODA dialog flow first verifies the channel specific OTP to authenticate the user.
- ODA then obtains token for the Siebel client registered in IDCS without any further inputs from user
- The token obtained is sent to Siebel API for any resource requests



IDCS Configuration

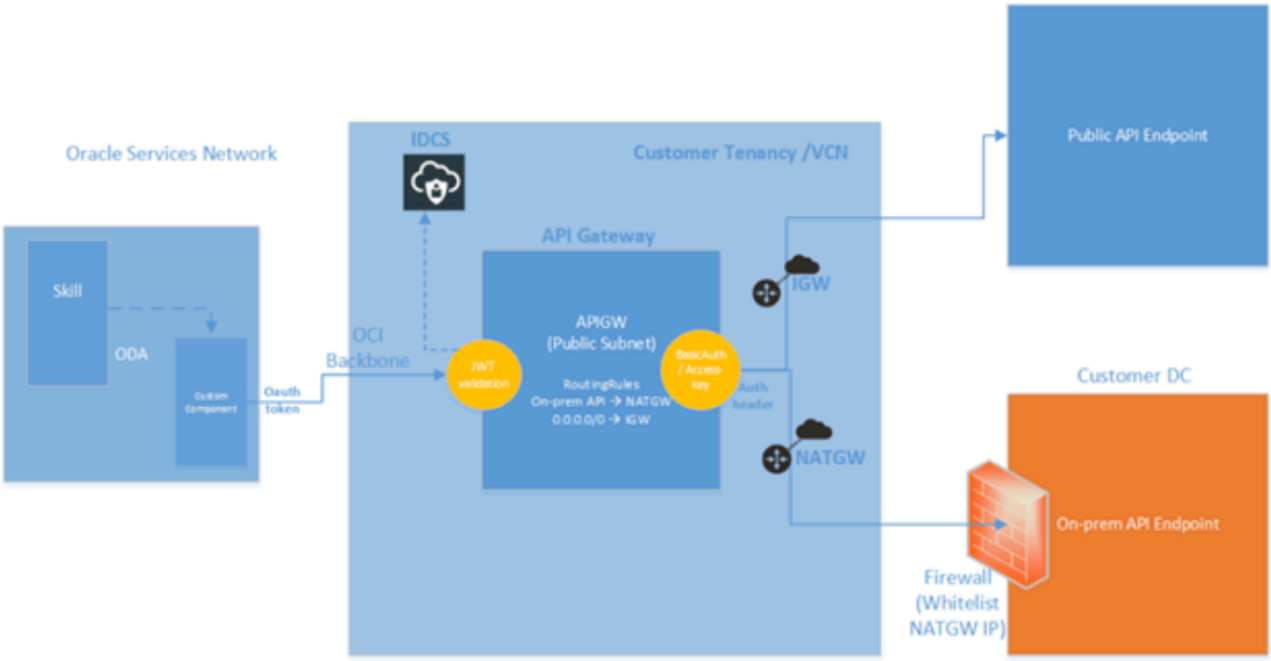
To configure IDCS, follow the below steps:

1. Login to IDCS admin console.
2. Go to Applications tab and click on "Add" and select "Confidential Application".
3. In the "Details" section, provide Name and Description and click on next.
4. Select "Configure this application as a client now ".
5. Select the following option in Authorization section and click on Next:
    a. Allowed Grant Types: Resource Owner, Client Credentials, Refresh Token, Authorization Code, Implicit
    b. Redirect URL: <ODA Callback URL>
    c. Allowed Operations: Introspect, On Behalf Of
6. Select "Configure this application as resource server now" and provide "Primary Audience" as "SEBL".
7. Click on "Add" button next to Scope and provide following scope:
    a. service
    b. data
8. Click on Next -> "Skip for Later" -> Next -> Finish. Click on Save.
9. Go to Applications again and drill down on your application. In the Configuration tab, click on Client Configurations. In that section under Resources, click on Add Scope.
10. From the popup that comes up, select your application and select both the scopes i.e. service and data, and Click Add.
11. In the Users tab, assign Users. If the user is not present, create a new user in Users section and then assign it to this application.
12. Activate this application.
13. Note down the client id and client secret from Configurations tab.

Connecting to Siebel on-premise

Security

- API Gateway endpoint is protected by token validation using default IDCS instance as the OAuth Auth server.
- ODA uses System.OAuth2Client Builtin component to retrieve Oauth access token from IDCS.
- ODA custom component passes the Oauth access token in its request to API gateway endpoint.
- API gateway strips off the Auth header after token is validated against IDCS.
- API gateway injects required basic auth or access-key credentials in the outbound request to on-prem backend endpoint.



7. Learn More

Below are some references that will further help you to develop your own Skills and use them in DA.

Using Oracle Digital Assistant: https://docs.oracle.com/en/cloud/paas/digital-assistant/use-chatbot/overview-digital-assistants-and-skills.html

Build Your First Skill With Oracle Digital Assistant: https://docs.oracle.com/en/cloud/paas/digital-assistant/tutorial-skill/index.html#create-order

Integrating Oracle Digital Assistant (ODA) with an Agent System: https://github.com/oracle/cloud-asset-oda-agent-handover

Siebel REST API Guide: https://docs.oracle.com/cd/F26413_21/books/RestAPI/index.html