Oracle® Cloud

Using the OpenAI Adapter with Oracle Integration 3

G36002-02
October 2025

Oracle Cloud Using the OpenAI Adapter with Oracle Integration 3,

G36002-02

Primary Author: Oracle Corporation

Contents

About This Content

This guide describes how to configure this adapter as a connection in an integration in Oracle Integration.

Audience

This guide is intended for developers who want to use this adapter in integrations in Oracle Integration.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Resources

See these Oracle resources:

- Oracle Cloud at `http://cloud.oracle.com`
- *Using Integrations in Oracle Integration 3*
- *Using the Oracle Mapper with Oracle Integration 3*
- Oracle Integration documentation on the Oracle Help Center.

Conventions

The following text conventions are used in this document.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understand the OpenAI Adapter

Review the following topics to learn about the OpenAI Adapter and how to use it as a connection in integrations in Oracle Integration. A typical workflow of adapter and integration tasks is also provided.

Topics:

OpenAI Adapter Capabilities

The OpenAI Adapter enables you to call large language models (LLMs) from integrations in Oracle Integration. The OpenAI APIs provide a simple REST interface to state-of-the-art LLM models for text generation, prompting, and function calling. Depending on the OpenAI API you use, you can call OpenAI, Anthropic, and Azure LLMs with the OpenAI Adapter.

The OpenAI Adapter provides the following capabilities:

- Supports the OpenAI Responses API and Chat Completions API. See Chat Completions and Responses. Each API offers different capabilities for calling LLMs:
  - The Responses API lets you call OpenAI LLMs.
  - The Chat Completions API lets you call OpenAI, Anthropic, and Azure LLMs.

  You select the API to use when configuring the OpenAI Adapter in the Adapter Endpoint Configuration Wizard. See Basic Info Page.
- Simple and extended prompt-based interactions to define the structure of the input: You provide instructions, questions, or plain text to generate contextual and coherent AI responses from a prompt. The text response can be code, mathematical equations, structured JSON data, or human-like prose.
- Role and context definitions: The following roles influence how the model responds and behaves.
  - `platform`: Messages added by OpenAI. These roles carry the highest authority and override all other instructions.
  - `developer`: Input from the Oracle Integration application developer.
  - `user`: Input from end users, or a catch-all for data provided as input to the model.
  - `assistant`: Input sampled from the LLM model.
  - `tool`: Input generated by some program, such as code execution or an API call.
- Context maintenance: The previous response ID is used to support ongoing conversations and follow-up questions.

- Model selection and tuning: You select from available OpenAI models and fine-tune responses with parameters such as temperature.
- Flexible input options: Simple prompts or structured arrays containing role/content pairs are supported.
- Function calling support: OpenAI models can interface with code or external services.
- Use cases: A variety of use cases are supported, including the following:
  - Text analysis: Understand and extract insights from textual data (that is, analyze employee sentiment from appraisal or customer feedback from product reviews).
  - Text categorization: Automatically classify documents or resumes based on content and predefined rules.
  - Email drafting: Generate intelligent email responses for customer queries, HR communication, or marketing campaigns.
  - Translation: Translate customer support responses or documentation into multiple languages.
  - Summaries: Condense long documents or reports into digestible summaries for decision makers.

You can configure the OpenAI Adapter as an invoke connection in an integration in Oracle Integration. The OpenAI Adapter is one of many predefined adapters included with Oracle Integration. See the Adapters page in the Oracle Help Center.

OpenAI Adapter Restrictions

Note the following OpenAI Adapter restrictions.

- When you define the `tools` parameter in your JSON structure, you must manually convert the JSON format into string format to successfully pass it to the OpenAI model. The following example shows a sample request JSON structure specified in the REST Adapter trigger connection. Back slashes are used to convert the JSON format to string format.

```
{ "previous_response_id" :
"resp_67fcf9d44a308136841b47ff252cb2c40f3934e820b8b734",
"messages" : [ { "role" : "user", "content" : "What is the weather like in
Boston today?", "id" :
"fc_68186d15c16c8192bfd40ea203016e2709a262500b6fb5c2", "type" :
"function_call", "status" :
"completed", "arguments" : "{\"document_name\":\"R010\"}", "call_id" :
"call_MUbVuvEfzFkwFwD30ZiBQCUg",
"name" : "classify_document", "output" : "receipts" }, { "role" : "user",
"content" :
"What is the weather like in Boston today?", "id" :
"fc_68186d15c16c8192bfd40ea203006e2709a262500b6fb5c2",
"type" : "function_call", "status" : "completed", "arguments" :
"{\"document_name\":\"R010\"}", "call_id" :
"call_MUbVuvEfzFkwFwD30ZiBQCUg", "name" : "classify_document", "output" :
"receipts" } ],
"tools" : "[ {\r\n \"type\" : \"function\",\r\n \"name\" :
\"classify_document\",\r\n
\"description\" : \"classifies document whether its item reciept or
invoice\",\r\n
\"parameters\" : {\r\n \"type\" : \"object\",\r\n \"required\" :
[ \"document_name\"],\r\n
```

```
\"properties\" : {\r\n \"document_name\" : {\r\n \"type\" :
\"string\",\r\n \"description\" :
\"This tells the document name\"\r\n }\r\n }\r\n }\r\n }, {\r\n \"type\" :
\"function\",\r\n \"name\" :
\"extract_receipt_data\",\r\n \"description\" : \"Extracts Receipt
data\",\r\n \"parameters\" : {\r\n
\"type\" : \"object\",\r\n \"required\" : [ \"receipt_id\"],\r\n
\"properties\" : {\r\n
\"receipt_id\" : {\r\n \"type\" : \"string\",\r\n \"description\" : \"the
receipt id to extract.\"\r\n }
\r\n }\r\n }\r\n }, {\r\n \"type\" : \"function\",\r\n \"name\" :
\"rag_retrieval\",\r\n \"description\" :
\"Extracts Receipt data\",\r\n \"parameters\" : {\r\n \"type\" :
\"object\",\r\n \"properties\" : {\r\n
\"question\" : {\r\n \"type\" : \"string\",\r\n \"description\" : \"The
user's query.\"\r\n },\r\n
\"collection_name\" : {\r\n \"type\" : \"string\",\r\n \"description\" :
\"Name of the ChromaDB collection.
\",\r\n \"default\" : \"oracle_integration_documents\",\r\n \"enum\" :
[ \"oracle_integration_documents\",
\"expense_report_policy_documents\" ]\r\n }\r\n },\r\n \"required\" :
[ \"question\"]\r\n }\r\n }, {\r\n
\"type\" : \"function\",\r\n \"name\" : \"create_expense_report\",\r\n
\"description\" : \"This tool creates
expense report\", \r\n \"parameters\" : {\r\n \"type\" : \"object\",\r\n
\"properties\" : {\r\n \"amount\" :
{\r\n \"type\" : \"string\",\r\n \"description\" : \"recipt expense
amount\"\r\n }\r\n },\r\n \"required\" :
[ \"amount\" ]\r\n }\r\n } ]" }
```

> **Note**
>
> There are overall service limits for Oracle Integration. A service limit is the quota or allowance set on a resource. See Service Limits.

What Application Version Is Supported?

For information about which application version is supported by this adapter, see the Connectivity Certification Matrix.

Workflow to Create and Add an OpenAI Adapter Connection to an Integration

You follow a very simple workflow to create a connection with an adapter and include the connection in an integration in Oracle Integration.

This table lists the workflow steps for both adapter tasks and overall integration tasks, and provides links to instructions for each step.

Step	Description	More Information
1	Decide where to work	• Work in a project (see why working with projects is preferred in *Using Integrations in Oracle Integration 3*). • Work outside a project.
2	Create the adapter connections for the applications you want to integrate. The connections can be reused in multiple integrations and are typically created by the administrator.	Create an OpenAI Adapter Connection
3	Create the integration. When you do this, you add trigger (source) and invoke (target) connections to the integration.	Understand Integration Creation and Best Practices in *Using Integrations in Oracle Integration 3* and Add the OpenAI Adapter Connection to an Integration
4	Map data between the trigger connection data structure and the invoke connection data structure.	Map Data in *Using Integrations in Oracle Integration 3*
5	(Optional) Create lookups that map the different values used by those applications to identify the same type of object (such as gender codes or country codes).	Manage Lookups in *Using Integrations in Oracle Integration 3*
6	Activate the integration.	Activate an Integration in *Using Integrations in Oracle Integration 3*
7	Monitor the integration on the dashboard.	Monitor Integrations During Runtime in *Using Integrations in Oracle Integration 3*
8	Track payload fields in messages during runtime.	Assign Business Identifiers for Tracking Fields in Messages and Track Integration Instances in *Using Integrations in Oracle Integration 3*
9	Manage errors at the integration level, connection level, or specific integration instance level.	Manage Errors in *Using Integrations in Oracle Integration 3*

2

Create an OpenAI Adapter Connection

A connection is based on an adapter. You define connections to the specific cloud applications that you want to integrate.

Topics:

Prerequisites for Creating a Connection

You must satisfy the following prerequisites to create a connection with the OpenAI Adapter:

- Purchase an API key. The key to purchase is based on the LLMs you want to call:
  - For OpenAI, see OpenAI Platform.
  - For Anthropic, see Build with Claude.
  - For Azure, see Azure AI Foundry.

  You need the key for configuring your OpenAI Adapter connection on the Connections page. See Configure Connection Security.
- Review the models to identify the ones that best serve your business needs. During OpenAI Adapter configuration in an integration, you must select the model to use. See:
  - OpenAI Platform Models
  - Claude Docs (Anthropic)
  - Azure AI Foundry Models

Create a Connection

Before you can build an integration, you must create the connections to the applications with which you want to share data.

> **Note**
>
> You can also create a connection in the integration canvas. See Define Inbound Triggers, Outbound Invokes, and Actions.

To create a connection in Oracle Integration:

1. Decide where to start:
   - Work in a project (see why working with projects is preferred).
     a. In the navigation pane, click **Projects**.
     b. Select the project name.

c. Click **Integrations**.

d. In the **Connections** section, click **Add** if no connections currently exist or **+** if connections already exist. The Create connection panel opens.

- Work outside a project.

a. In the navigation pane, click **Design**, then **Connections**.

b. Click **Create**. The Create connection panel opens.

2. Select the adapter to use for this connection. To find the adapter, scroll through the list, or enter a partial or full name in the **Search** field.
3. Enter the information that describes this connection.

Element	Description
Name	Enter a meaningful name to help others find your connection when they begin to create their own integrations.
Identifier	Automatically displays the name in capital letters that you entered in the **Name** field. If you modify the identifier name, don't include blank spaces (for example, `SALES_OPPORTUNITY`).
Role	Select the role (direction) in which to use this connection. **Note**: *Only* the roles supported by the adapter you selected are displayed for selection. Some adapters support all role combinations (trigger, invoke, or trigger and invoke). Other adapters support fewer role combinations. When you select a role, only the connection properties and security policies appropriate to that role are displayed on the Connections page. If you select an adapter that supports both invoke and trigger, but select only one of those roles, you'll get an error when you try to drag the adapter into the section you didn't select. For example, assume you configure a connection for the Oracle Service Cloud (RightNow) Adapter as only an **invoke**. Dragging the adapter to a **trigger** section in the integration produces an error.
Keywords	Enter optional keywords (tags). You can search on the connection keywords on the Connections page.
Description	Enter an optional description of the connection.

Element	Description
Share with other projects	**Note**: This field only appears if you are creating a connection in a project. Select to make this connection publicly available in other projects. Connection sharing eliminates the need to create and maintain separate connections in different projects. When you configure an adapter connection in a different project, the **Use a shared connection** field is displayed at the top of the Connections page. If the connection you are configuring matches the same type and role as the publicly available connection, you can select that connection to reference (inherit) its resources. See Add and Share a Connection Across a Project.

4. Click **Create**.

   Your connection is created. You're now ready to configure the connection properties, security policies, and (for some connections) access type.

5. Follow the steps to configure a connection.
   The connection property and connection security values are specific to each adapter. Your connection may also require configuration with an access type such as a private endpoint or an agent group.

6. Test the connection.

Configure Connection Properties

Enter connection information so your application can process requests.

1. Go to the **Properties** section.
2. Enter the following information.

Element	Description

Base URL	Enter the base URL for the LLMs you want to use.
	The OpenAI Adapter supports the Responses API and the Chat Completions API. The URL to enter in the **Base URL** field is based on the API you want to use. You select the API to use on the Basic Info page of the Adapter Endpoint Configuration Wizard.
	• If you use the Responses API, you can only call OpenAI LLMs.
	• If you use the Chat Completions API, you can call OpenAI, Anthropic, and Azure LLMs.
	To call OpenAI models:
	`https://api.openai.com`
	To call Anthropic models:
	`https://api.anthropic.com`
	To call Azure models:
	`https://ai.azure.com`

Model	Enter the LLM you want to use.

Configure Connection Security

Configure security for your OpenAI Adapter connection.

1. Go to the **Security** section.
2. In the **API Key Based Authentication** field, enter the API key you obtained from OpenAI, Anthropic, or Azure. The API key is used for authentication.

Test the Connection

Test your connection to ensure that it's configured successfully.

1. In the page title bar, click **Test**. What happens next depends on whether your adapter connection uses a Web Services Description Language (WSDL) file. Only some adapter connections use WSDLs.

If Your Connection...	Then...
Doesn't use a WSDL	The test starts automatically and validates the inputs you provided for the connection.
Uses a WSDL	A dialog prompts you to select the type of connection testing to perform: • **Validate and Test**: Performs a full validation of the WSDL, including processing of the imported schemas and WSDLs. Complete validation can take several minutes depending on the number of imported schemas and WSDLs. No requests are sent to the operations exposed in the WSDL. • **Test**: Connects to the WSDL URL and performs a syntax check on the WSDL. No requests are sent to the operations exposed in the WSDL.

2. Wait for a message about the results of the connection test.
   - If the test was successful, then the connection is configured properly.
   - If the test failed, then edit the configuration details you entered. Check for typos and verify URLs and credentials. Continue to test until the connection is successful.
3. When complete, click **Save**.

3

Add the OpenAI Adapter Connection to an Integration

When you drag the OpenAI Adapter into the invoke area of an integration, the Adapter Endpoint Configuration Wizard is invoked. This wizard guides you through configuration of the OpenAI Adapter endpoint properties.

The following sections describe the wizard pages that guide you through configuration of the OpenAI Adapter as an invoke in an integration.

Topics:

Basic Info Page

Specify a name, description, and subscription type on the Basic Info page of each trigger connection in your integration.

Element	Description
What do you want to call your endpoint?	Provide a meaningful name so that others can understand the connection. For example, if you are creating a database connection for adding new employee data, you may want to name it `CreateEmployeeInDB`. You can include English alphabetic characters, numbers, underscores, and dashes in the name. You cannot include the following: • Blank spaces (for example, `My DB Connection`) • Special characters (for example, `#;83&` or `righ(t)now4`) except underscores and hyphens • Multibyte characters
What does this endpoint do?	Enter an optional description of the connection's responsibilities.
Action	Select the API to use in integrations. • **Responses API**: Combines the simplicity of the older Chat Completions APIs with the ability to do more agentic tasks. The Responses API provides a foundation for building action-oriented applications, with built-in tools such as Web search, File search, and Computer use. You can call OpenAI LLMs with this API. The Responses API is recommended for new users. • **Chat completions API**: Specifically designed to support conversational (chat) experiences and more advanced use cases such as virtual assistants, chat bots, and intelligent conversation agents. You can call OpenAI, Anthropic, and Azure LLMs with this API.

Invoke Configuration Page

Select the OpenAI model to use and define the structure of the input request (either a simple prompt or an extended prompt). This page is only displayed if you selected the **Responses API** action on the Basic Info page.

Element	Description
OpenAI LLM Models	Select the model to use in this integration. See OpenAI Platform Models.

Request Type	Select the request type to define the structure of the input request:
	• **Simple Prompt**: Enables you to provide a straightforward instruction or question to the OpenAI model. For example:
	`"input": "In Which country is San Francisco located"`
	The single prompt focuses on clarity and conciseness, avoiding complex phrasing or unnecessary details that can potentially confuse the model.
	• **Extended Prompt**: Enables you to provide a specific implementation that uses multiple roles and the OpenAI API function calling feature. The following example shows the use of multiple roles:
	`"input": [{ "role": "developer", "content": "Give information only about Boston" }, { "role": "user", "content": "What is the zipcode of Beacon Hill, Boston?"`
	This framework allows the model to interact with services and perform actions based on your prompt.
	See Implement Common Patterns Using the OpenAI Adapter.

Summary Page

You can review the specified adapter configuration values on the Summary page.

Element	Description
Summary	Displays a summary of the configuration values you defined on previous pages of the wizard. The information that is displayed can vary by adapter. For some adapters, the selected business objects and operation name are displayed. For adapters for which a generated XSD file is provided, click the XSD link to view a read-only version of the file. To return to a previous page to update any values, click the appropriate tab in the left panel or click **Go back**. To cancel your configuration details, click **Cancel**.

4

Implement Common Patterns Using the OpenAI Adapter

You can use the OpenAI Adapter to implement the following common patterns.

Topics:

Provide a Simple Instruction to the OpenAI Model

This use case demonstrates how to provide a simple question to the specified OpenAI model when using the Responses API. The simple prompt focuses on clarity and conciseness, avoiding complex phrasing or unnecessary details that can potentially confuse the model.

1. Configure a REST Adapter trigger connection.
2. Configure an OpenAI Adapter invoke connection. See Create a Connection.
3. Create an application integration.
4. Drag the REST Adapter trigger connection into the integration canvas for configuration. For this example, the REST Adapter is configured as follows:
   - A **REST Service URL** of `/extended` is specified for this example.
   - A **Method** of **POST** is selected.
   - A **Request Media Type** of **JSON** is selected and the following sample JSON structure is specified:

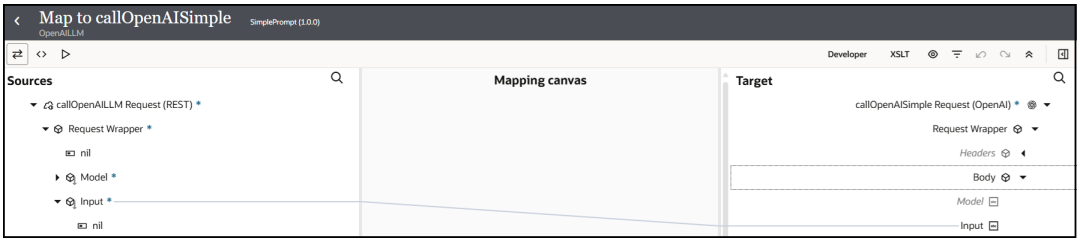
     ```
     { "model" : "gpt-4o", "input" : "register the functions for tool
     calling"
         }
     ```

   - A **Response Media Type** of **JSON** is selected and the following sample JSON structure is specified:

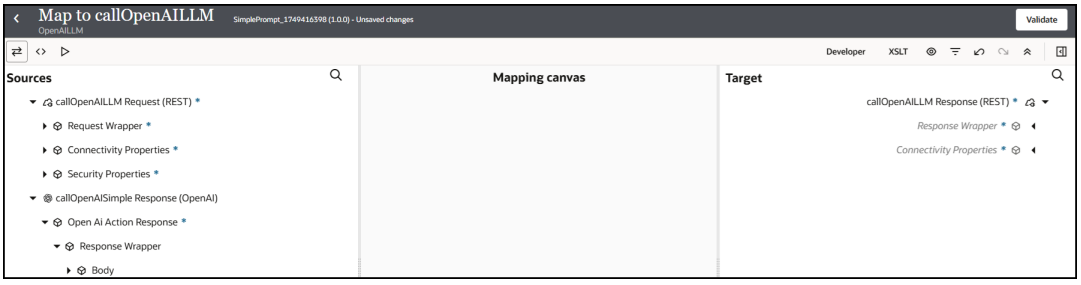
     ```
     { "id" : "resp_682b6c897b408198aff19b602ca3f0a20b404da49e82c3e4",
     "object" : "response",
     "created_at" : 1747676297, "status" : "completed", "model" :
     "gpt-4o-2024-08-06", "output" :
     [ { "id" : "msg_682b6c8a1fd08198b585adaae3f27b6e0b404da49e82c3e4",
     "type" : "message", "status" :
     "completed", "content" : [ { "type" : "output_text", "text" : "Welcome!
     How can I assist you today?" } ],
     "role" : "assistant" } ], "parallel_tool_calls" : true,
     ```

```
"previous_response_id" : "abcd", "reasoning" :
{ "effort" : "abcd", "summary" : "abcd" }, "service_tier" : "default",
"store" : true, "temperature" : 1.0,
"text" : { "format" : { "type" : "text" } }, "tool_choice" : "auto",
"top_p" : 1.0, "truncation" : "disabled",
"usage" : { "input_tokens" : 12, "input_tokens_details" :
{ "cached_tokens" : 0 }, "output_tokens" : 10,
"output_tokens_details" : { "reasoning_tokens" : 0 }, "total_tokens" :
22 } }
```

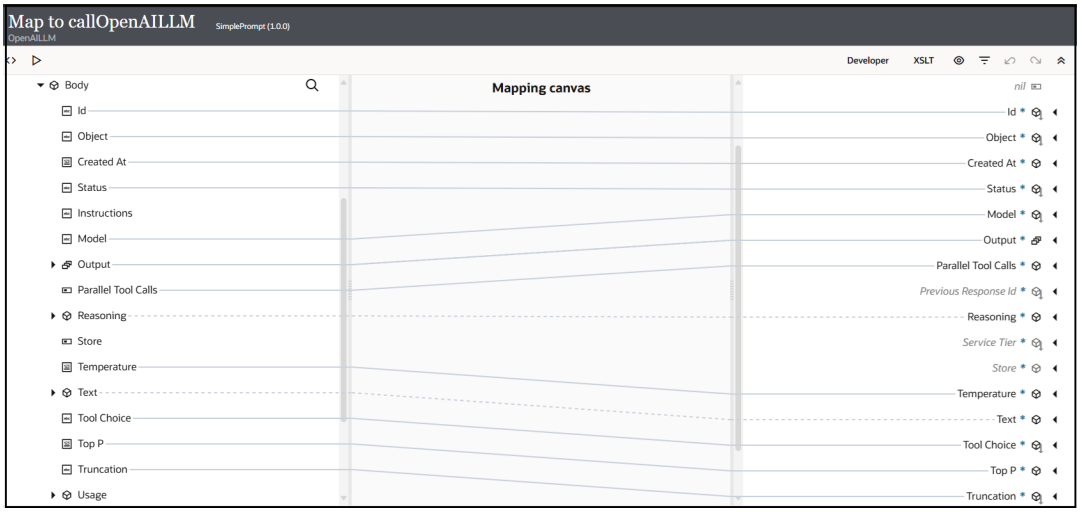

5. Drag the OpenAI Adapter invoke connection into the integration canvas and configure it as follows.
   a. On the Basic Info page, select **Responses API** from the **Action** list.
   b. On the Configuration page, select the following:
      i. From the **OpenAI LLM Models** list, select the model to use (for this example, **gpt-4o** is selected).
      ii. From the **Request Type** list, select **Simple Prompt**.
6. Open the request mapper automatically created when the OpenAI Adapter invoke connection was added to the integration. The mapper that was automatically created with the REST Adapter trigger connection is not edited and remains empty in this use case.
7. In the request mapper, map the source **Input** element to the target **Input** element.



8. In the response mapper, expand the source **Body** element and target **Response Wrapper** element.

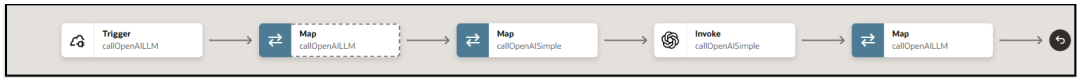


9. Perform the following source-to-target mappings.



10. Specify a business identifier and activate the integration.

    The completed integration looks as follows:



11. From the **Actions** ⋯ menu, select **Run**.

    The Configure and run page appears.

12. In the **Body** field of the **Request** section, enter the following content, then click **Run**.

    - `input`: Enter the text input to the model.
    - `model`: Specify the model ID to generate the response. This is the model you selected when configuring the OpenAI Adapter in the integration.

```
{
  "input": "In Which country is San Francisco located",
  "model": "gpt-4o"
}
```

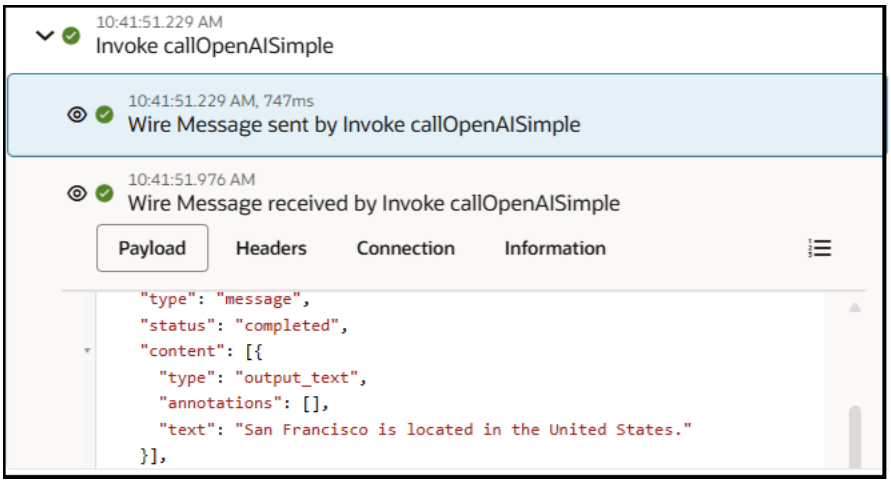
The **Body** field of the **Response** section returns the following output. The country in which San Francisco is located is returned.

```
{
  "id" : "resp_6845cb5f623881998e1652ac9d60b669087e7c84ef2dd435",
  "object" : "response",
  "created_at" : 1749404511,
  "status" : "completed",
  "model" : "gpt-4o-2024-08-06",
  "output" : [ {
    "id" : "msg_6845cb5fcc0c81998c5e8fe27d92ff66087e7c84ef2dd435",
    "type" : "message",
    "content" : [ {
      "type" : "output_text",
      "text" : "San Francisco is located in the United States."
    } ]
```

```
  } ],
  "parallel_tool_calls" : true,
  "reasoning" : {
    "effort" : ""
  },
  "temperature" : 1,
  "text" : {
    "format" : {
      "type" : "text"
    }
  },
  "tool_choice" : "auto",
  "top_p" : 1,
  "truncation" : "disabled"
}
```

13. Expand the activity stream to view the flow of the messages sent and received.
    - Message sent by the invoke connection to the OpenAI model:

    - Message received by the invoke connection from the OpenAI model:



Provide an Extended Instruction to the OpenAI Model

This use case demonstrates how to provide an array-based instruction to the OpenAI model when using the Responses API. Two roles are specified in the input. For each role, you define different content for the OpenAI model to address.

1. Configure a REST Adapter trigger connection.
2. Configure an OpenAI Adapter invoke connection.
3. Create an application integration.
4. Drag the REST Adapter trigger connection into the integration canvas for configuration. For this example, the REST Adapter is configured as follows:
   - A **REST Service URL** of `/extended3` is specified for this example.
   - A **Method** of **POST** is selected.
   - A **Request Media Type** of **JSON** is selected and the following sample JSON structure is specified:

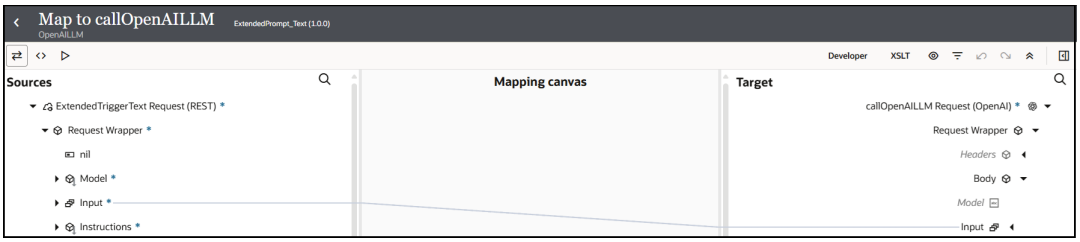
     ```
     { "model" : "gpt-4o", "input" : [ { "role" : "developer", "content" :
     "perform the openAI LLM function calling functionality described in the
     form of
     text content. Response should be same as that returned for function
     calling" },
     { "role" : "user", "content" : "Define a get_weather function with
     parameters
     location, which is a string object. Call the appropriate function for
     What is
     the weather like in Paris today?" } ], "instructions" : "",
     "max_output_tokens" : 234,
     "metadata" : null, "parallel_tool_calls" : true,
     "previous_response_id" : null,
      "store" : true, "stream" : false, "temperature" : 1, "tool_choice" :
     "auto",
     "top_p" : 1, "truncation" : "disabled", "user" : "asdf" }
     ```

   - A **Response Media Type** of **JSON** is selected and the following sample JSON structure is specified:

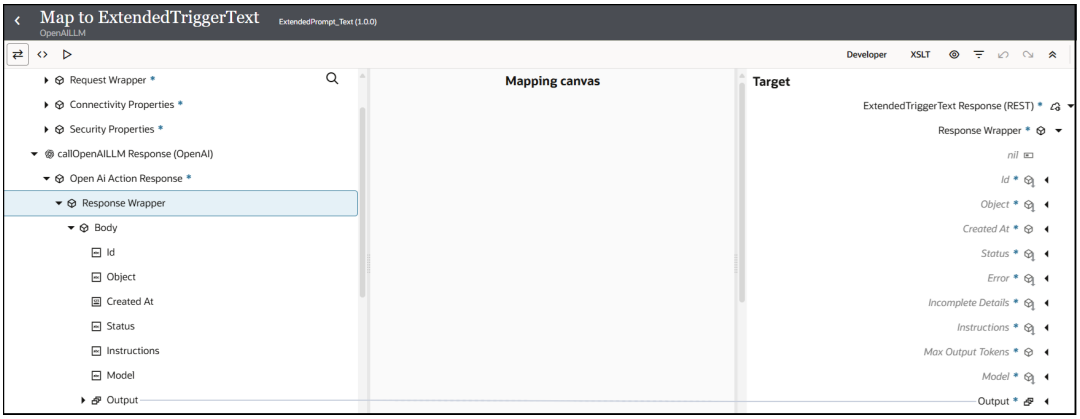
     ```
     { "id" : "resp_67e6322ad4688192abad3f268b66236e05ecd4d8d90549f9",
     "object" : "response",
     "created_at" : 1743139370, "status" : "completed", "error" : "df",
     "incomplete_details" : "asdf",
     "instructions" : "asdf", "max_output_tokens" : 243, "model" :
     "gpt-4o-2024-08-06", "output" : [ { "type" :
     "message", "id" :
     "msg_67e6322b18c08192a6352151edd8c9fa05ecd4d8d90549f9", "status" :
     "completed", "role" :
     "assistant", "content" : [ { "type" : "output_text", "text" : "Get
     current temperature for a given location." } ] } ],
     "parallel_tool_calls" : true, "previous_response_id" : "afsd",
     "reasoning" : { "effort" : "sdaf", "generate_summary" :
     "asf" }, "store" : true, "temperature" : 1, "text" : { "format" :
     { "type" : "text" } }, "tool_choice" : "auto", "top_p" : 1,
     ```

```
"truncation" : "disabled", "usage" : { "input_tokens" : 43,
"input_tokens_details" : { "cached_tokens" : 0 }, "output_tokens" :
68, "output_tokens_details" : { "reasoning_tokens" : 0 },
"total_tokens" : 111 }, "user" : "asdf" }
```

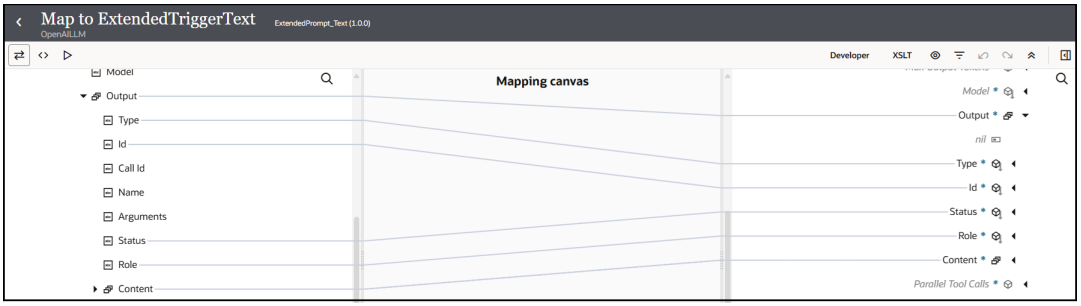
5. Drag the OpenAI Adapter invoke connection into the integration canvas and configure it as follows.
   a. On the Basic Info page, select **Responses API** from the **Action** list.
   b. On the Configuration page, select the following:
      i. From the **OpenAI LLM Models** list, select the model to use (for this example, **gpt-4o** is selected).
      ii. From the **Request Type** list, select **Extended Prompt**.
6. Open the request mapper automatically created when the OpenAI Adapter invoke connection was added to the integration. The mapper that was automatically created with the REST Adapter trigger connection is not edited and remains empty in this use case.
7. In the request mapper, map the source **Input** element to the target **Input** element.



8. In the response mapper, expand the source **Response Wrapper** element and target **Response Wrapper** element.

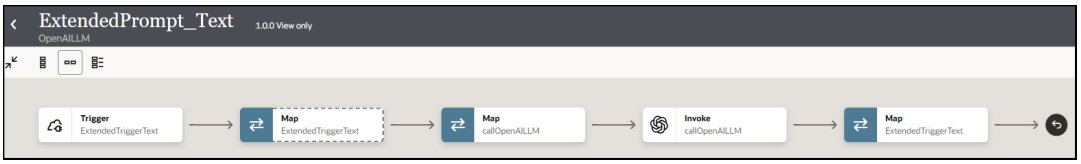


9. Perform the following mappings.



10. Specify the business identifier and activate the integration.

    The completed integration looks as follows:



11. From the **Actions** ... menu, select **Run**.

    The Configure and run page appears.

12. In the **Body** field of the **Request** section, enter the following text, then click **Run**.

    The body includes two roles (`developer` and `user`), each with their own content. The `developer` role takes precedence over the `user` role in the OpenAI hierarchy. For example, if you were to change the `user` role content from asking for the Boston zip code to asking for the zip code of a neighborhood in New York, the OpenAI model would not be able to answer the question.

```
{
"input":  [{
"role": "developer",
"content": "Give information only about Boston"
}, {
  "role": "user",
  "content": "What is the zipcode of Beacon Hill, Boston?"
}]
}
```

    The **Body** field of the **Response** section returns the following output. The zip code of Beacon Hill is returned.

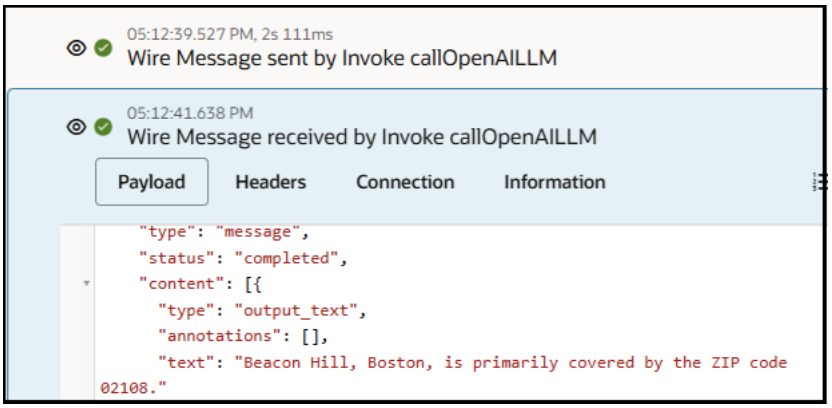
```
{
  "output" : [ {
    "type" : "message",
    "id" : "msg_68477879290c819890e84a6f557f0b560cec1aa24c1b96c8",
    "status" : "completed",
    "role" : "assistant",
    "content" : [ {
      "type" : "output_text",
      "text" : "Beacon Hill, Boston, is primarily covered by the ZIP code
02108."
```

```
    } ]
  } ]
}
```

13. Expand the activity stream to view the flow of the messages sent and received.
    - Message received by the trigger connection:

    - Message sent by the invoke connection to the OpenAI model:

    - Message received by the invoke connection from the OpenAI model:



Provide an Extended Instruction to the OpenAI Model to Use a Function

This use case demonstrates how to provide an array-based instruction to the OpenAI model when using the Responses API. Two roles are specified in the input. For each role, you define the content. The content for one of the roles instructs the OpenAI model to invoke a weather function.

This use case implements the same integration described in Provide an Extended Instruction to the OpenAI Model. The only difference is the request payload content that you specify on the Configure and run page at runtime.

1. Configure a REST Adapter trigger connection.
2. Configure an OpenAI Adapter invoke connection.
3. Create an application integration.
4. Drag the REST Adapter trigger connection into the integration canvas and configure. For this example, it is configured as follows:
   - A **REST Service URL** of `/extended3` is specified for this example.
   - A **Method** of **POST** is selected.
   - A **Request Media Type** of **JSON** is selected and the following sample JSON structure is specified:

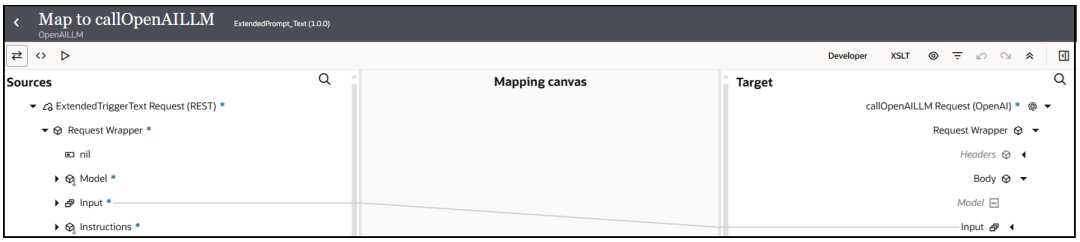
     ```
     { "model" : "gpt-4o", "input" : [ { "role" : "developer", "content" :
     "perform the openAI LLM function calling functionality described in the
     form of
     text content. Response should be same as that returned for function
     calling" },
     { "role" : "user", "content" : "Define a get_weather function with
     parameters
     location, which is a string object. Call the appropriate function for
     What is
     the weather like in Paris today?" } ], "instructions" : "",
     "max_output_tokens" : 234,
     "metadata" : null, "parallel_tool_calls" : true,
     "previous_response_id" : null,
      "store" : true, "stream" : false, "temperature" : 1, "tool_choice" :
     "auto",
     "top_p" : 1, "truncation" : "disabled", "user" : "asdf" }
     ```

   - A **Response Media Type** of **JSON** is selected and the following sample JSON structure is specified:

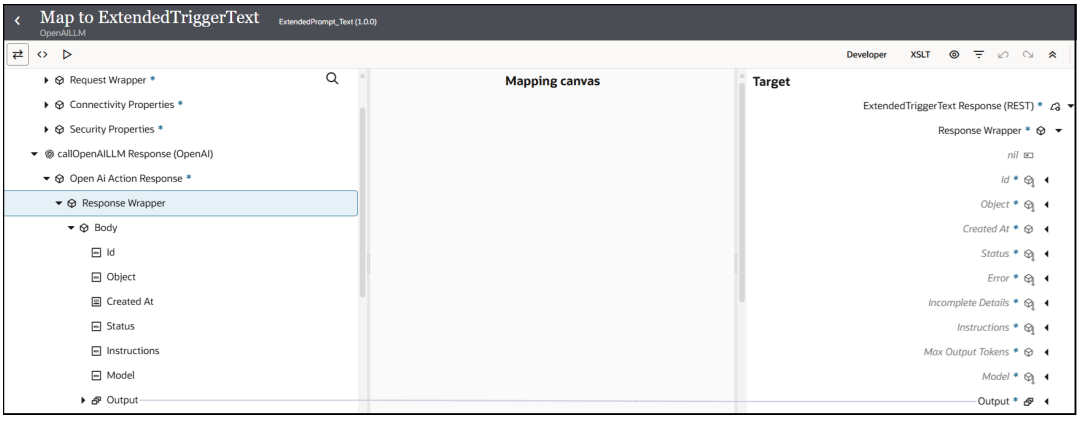
     ```
     { "id" : "resp_67e6322ad4688192abad3f268b66236e05ecd4d8d90549f9",
     "object" : "response",
     "created_at" : 1743139370, "status" : "completed", "error" : "df",
     "incomplete_details" : "asdf",
     "instructions" : "asdf", "max_output_tokens" : 243, "model" :
     "gpt-4o-2024-08-06", "output" : [ { "type" :
     "message", "id" :
     "msg_67e6322b18c08192a6352151edd8c9fa05ecd4d8d90549f9", "status" :
     "completed", "role" :
     ```

```
"assistant", "content" : [ { "type" : "output_text", "text" : "Get
current temperature for a given location." } ] } ],
"parallel_tool_calls" : true, "previous_response_id" : "afsd",
"reasoning" : { "effort" : "sdaf", "generate_summary" :
"asf" }, "store" : true, "temperature" : 1, "text" : { "format" :
{ "type" : "text" } }, "tool_choice" : "auto", "top_p" : 1,
"truncation" : "disabled", "usage" : { "input_tokens" : 43,
"input_tokens_details" : { "cached_tokens" : 0 }, "output_tokens" :
68, "output_tokens_details" : { "reasoning_tokens" : 0 },
"total_tokens" : 111 }, "user" : "asdf" }
```

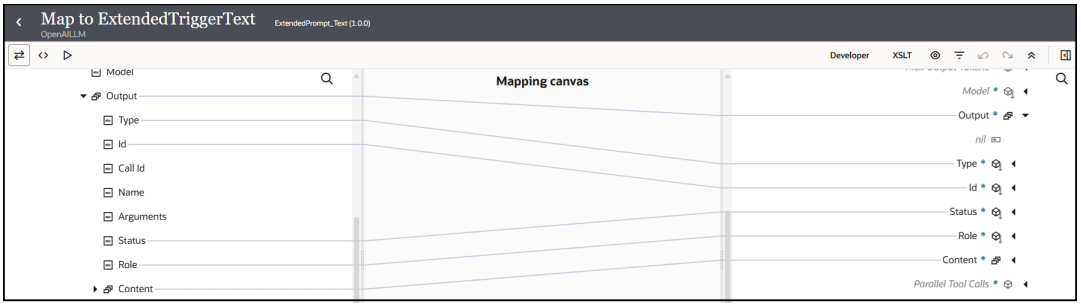
5. Drag the OpenAI Adapter invoke connection into the integration canvas and configure it as follows.
   a. On the Basic Info page, select **Responses API** from the **Action** list.
   b. On the Configuration page, select the following:
      i. From the **OpenAI LLM Models** list, select the model to use (for this example, **gpt-4o** is selected).
      ii. From the **Request Type** list, select **Extended Prompt**.
6. Open the request mapper automatically created when the OpenAI Adapter invoke connection was added to the integration. The mapper that was automatically created with the REST Adapter trigger connection is not edited and remains empty in this use case.
7. In the request mapper, map the source **Input** element to the target **Input** element.



8. In the response mapper, expand the source **Response Wrapper** element and target **Response Wrapper** element.

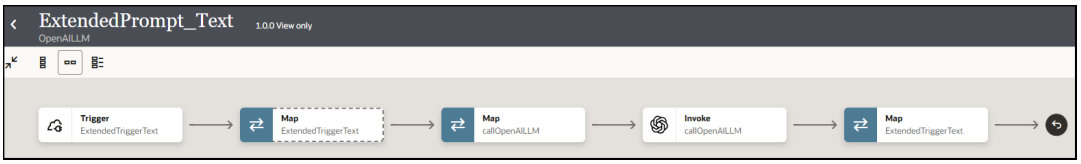


9. Perform the following mappings.



10. Specify the business identifier and activate the integration.

    The completed integration looks as follows:



11. From the **Actions** ... menu, select **Run**.

    The Configure and run page appears.

12. In the **Body** field of the **Request** section, enter the following text, then click **Run**.

    The body includes two roles (`developer` and `user`), each with their own content. The `user` content requests the OpenAI model to perform a `get_weather` function.

```
{
  "instructions": "",
  "metadata": null,
  "store": true,
  "top_p": 1,
  "input": [{
    "role": "developer",
    "content": "perform the openAI LLM function calling functionality
described
in the form of text content. Response should be same as that returned for
function
calling"
  }, {
    "role": "user",
    "content": "Define a get_weather function with parameters location,
which is a
string object. Call the appropriate function for What is the weather like
in Paris today?"
  }],
  "previous_response_id": null,
  "parallel_tool_calls": true,
  "stream": false,
  "temperature": 1,
  "tool_choice": "auto",
  "model": "gpt-4o",
  "truncation": "disabled",
  "user": "asdf",
```

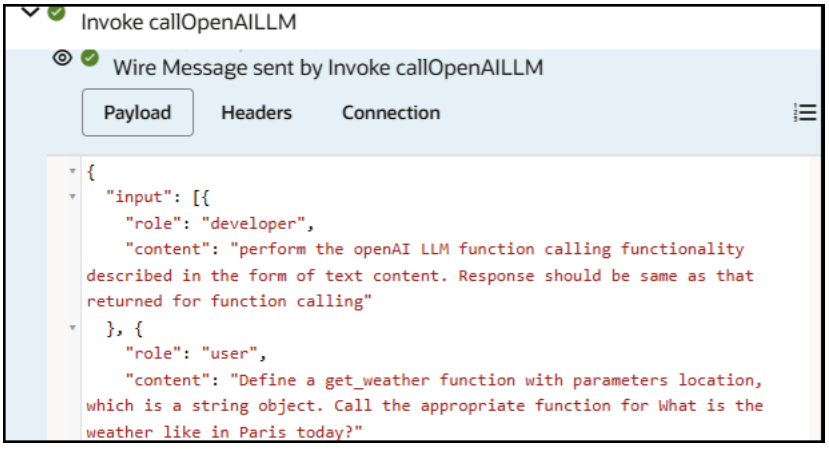
```
  "max_output_tokens": 234
}
```

The **Body** field of the **Response** section returns the following output:

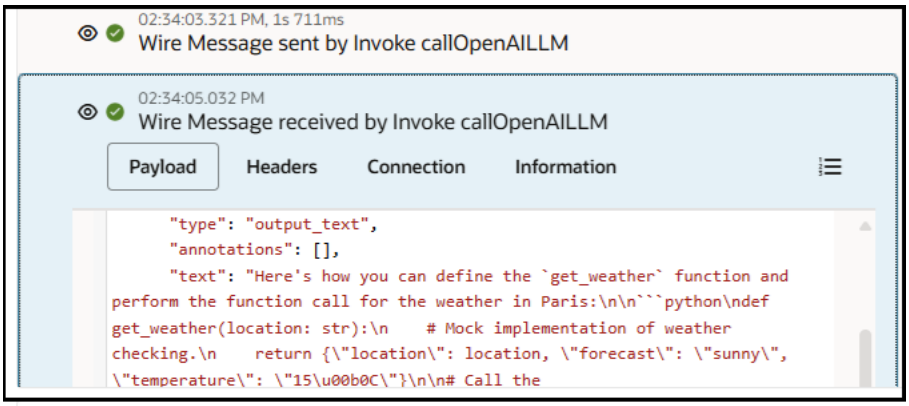
```
{
  "output" : [ {
    "type" : "message",
    "id" : "msg_684601cbd75c819b85e9067ac59631ec07e1a9c0b54af34a",
    "status" : "completed",
    "role" : "assistant",
    "content" : [ {
      "type" : "output_text",
      "text" : "Here's how you can define the `get_weather` function and
perform
the function call for the weather in Paris:\n\n```python\ndef
get_weather(location: str):\n    # Mock implementation of weather
checking.\n
return {\"location\": location, \"forecast\": \"sunny\", \"temperature\":
\"15°C\"}\n\n#
Call the function\nget_weather(\"Paris\")\n```\n\nFunction call
output:\n```json\n{\n
\"location\": \"Paris\",\n  \"forecast\": \"sunny\",\n  \"temperature\":
\"15°C\"\n}\n```"
    } ]
  } ]
}
```

13. Expand the activity stream to view the flow of the messages sent and received.
    - Message sent to the trigger connection:

    - Message sent by the invoke connection to the OpenAI model:



- Message received by the invoke connection from the OpenAI model:



- Message reply to the trigger connection:

Design an Integration Using the Chat Completions API

This use case demonstrates how to design an integration when using the Chat Completions API. The Chat Completions API enables you to call multiple LLMs (OpenAI, Anthropic, and

Azure). This use case also provides an overview of the mapper differences between the Chat Completions API and Responses API input and output profiles.

> **Note**
>
> The Chat Completions API makes no distinction between simple and extended prompts. By default, there is just an extended prompt. This differs from the Responses API, which provides both simple and extended prompts.

1. Configure a REST Adapter trigger connection.
2. Configure an OpenAI Adapter invoke connection.
3. Create an application integration.
4. Drag the REST Adapter trigger connection into the integration canvas and configure it. For this example, the adapter is configured as follows:
   - A **REST Service URL** of `/extended` is specified for this example.
   - A **Method** of **POST** is selected.
   - **Configure a request payload for this endpoint** and **Configure this endpoint to receive the response** are selected.
   - A **Request Media Type** of **JSON** is selected and the following sample JSON structure is specified:

```
{
  "previous_response_id" :
"resp_67fcf9d44a441592841b47ff452bb2c40f3934e820b8b734",
  "messages" : [ {
    "role" : "user",
    "content" : "What is the weather like in Boston today?",
    "id" : "fc_59976d15c16c8192bfd40ea202148e2709a262500b6fb5c2",
    "type" : "function_call",
    "status" : "completed",
    "arguments" : "{\"document_name\":\"R010\"}",
    "call_id" : "call_MUbVuvEfzFkwFwD30ZiBQCUg",
    "name" : "classify_document",
    "output" : "receipts"
  }, {
    "role" : "user",
    "content" : "What is the weather like in Boston today?",
    "id" : "fc_59976d15c16c8192bfd40ea202148e2709a262500b6fb5c2",
    "type" : "function_call",
    "status" : "completed",
    "arguments" : "{\"document_name\":\"R010\"}",
    "call_id" : "call_MUbVuvEfzFkwFwD30ZiBQCUg",
    "name" : "classify_document",
    "output" : "receipts"
  } ],
  "tools" : "[ {\r\n    \"type\" : \"function\",\r\n    \"name\" :
\"classify_document\",\r\n    \"description\" : \"classifies document
whether its item reciept or invoice\",\r\n    \"parameters\" :
{\r\n      \"type\" : \"object\",\r\n      \"required\" :
[ \"document_name\"],\r\n      \"properties\" : {\r\n
\"document_name\" : {\r\n          \"type\" : \"string\",\r\n
```

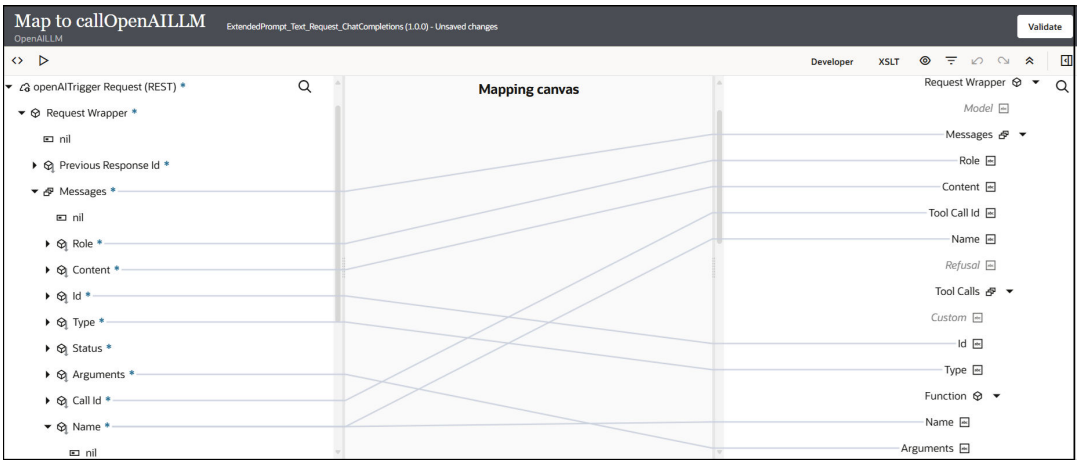
```
\"description\" : \"This tells the document name\"\r\n        }
\r\n      }\r\n    }\r\n  }, {\r\n    \"type\" : \"function\",\r\n
\"name\" : \"extract_receipt_data\",\r\n    \"description\" :
\"Extracts Receipt data\",\r\n    \"parameters\" : {\r\n
\"type\" : \"object\",\r\n      \"required\" :
[ \"receipt_id\"],\r\n      \"properties\" : {\r\n
\"receipt_id\" : {\r\n          \"type\" : \"string\",\r\n
\"description\" : \"the receipt id to extract.\"\r\n        }\r\n      }
\r\n    }\r\n  }, {\r\n    \"type\" : \"function\",\r\n    \"name\" :
\"rag_retrieval\",\r\n    \"description\" : \"Extracts Receipt
data\",\r\n    \"parameters\" : {\r\n      \"type\" :
\"object\",\r\n      \"properties\" : {\r\n        \"question\" :
{\r\n          \"type\" : \"string\",\r\n          \"description\" :
\"The user's query.\"\r\n        },\r\n        \"collection_name\" :
{\r\n          \"type\" : \"string\",\r\n          \"description\" :
\"Name of the ChromaDB collection.\",\r\n          \"default\" :
\"oracle_integration_documents\",\r\n          \"enum\" :
[ \"oracle_integration_documents\",
\"expense_report_policy_documents\" ]\r\n        }\r\n      },\r\n
\"required\" : [ \"question\"]\r\n    }\r\n  }, {\r\n    \"type\" :
\"function\",\r\n    \"name\" : \"create_expense_report\",\r\n
\"description\" : \"This tool creates expense report\",\r\n
\"parameters\" : {\r\n      \"type\" : \"object\",\r\n
\"properties\" : {\r\n        \"amount\" : {\r\n          \"type\" :
\"string\",\r\n          \"description\" : \"recipt expense
amount\"\r\n        }\r\n      },\r\n      \"required\" : [ \"amount\" ]
\r\n    }\r\n  } ]"
}
```

- A **Response Media Type** of **JSON** is selected and the following sample JSON structure is specified:

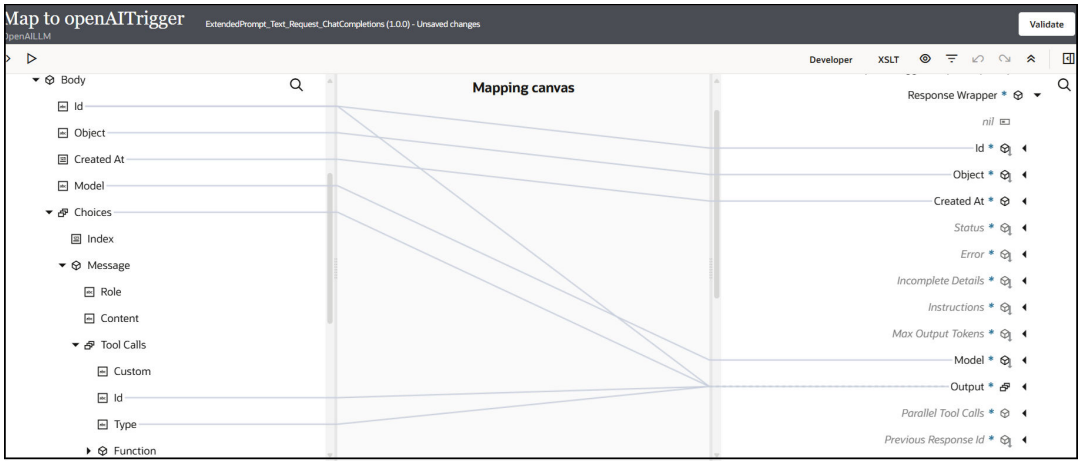
```
{
  "id" : "resp_68109ae7dd748192a3b562b362fa7e3104b7fafa5b45e5a9",
  "object" : "response",
  "created_at" : 1745918695,
  "status" : "completed",
  "error" : null,
  "incomplete_details" : null,
  "instructions" : null,
  "max_output_tokens" : null,
  "model" : "gpt-3.5-turbo-0125",
  "output" : [ {
    "id" : "fc_68109aed4f98819288179 69dfa1f0fcc04b7fafa5b45e5a9",
    "type" : "function_call",
    "status" : "completed",
    "arguments" : "{\"location\":\"Boston, MA\",\"unit\":\"celsius\"}",
    "call_id" : "call_C65f116qkbd246rHQAnftPcf",
    "name" : "get_current_weather"
  } ],
  "parallel_tool_calls" : true,
  "previous_response_id" : null,
  "reasoning" : {
    "effort" : null,
    "summary" : null
  },
```

```
  "service_tier" : "default",
  "store" : true,
  "temperature" : 1.0,
  "text" : {
    "format" : {
      "type" : "text"
    }
  },
  "tool_choice" : "auto",
  "tools" : [ {
    "type" : "function",
    "name" : "classify_document",
    "description" : "classifies document whether its item reciept or
invoice",
    "parameters" : {
      "type" : "object",
      "properties" : {
        "document_name" : {
          "type" : "string",
          "description" : "This tells the document name"
        }
      },
      "required" : [ "document_name" ]
    },
    "strict" : true
  }, {
    "type" : "function",
    "name" : "extract_receipt_data",
    "description" : "Extracts Receipt data",
    "parameters" : {
      "type" : "object",
      "properties" : {
        "receipt_id" : {
          "type" : "string",
          "description" : "the receipt id to extract."
        }
      },
      "required" : [ "receipt_id" ]
    },
    "strict" : true
  } ],
  "top_p" : 1.0,
  "truncation" : "disabled",
  "usage" : {
    "input_tokens" : 86,
    "input_tokens_details" : {
      "cached_tokens" : 0
    },
    "output_tokens" : 23,
    "output_tokens_details" : {
      "reasoning_tokens" : 0
    },
    "total_tokens" : 109
  },
  "user" : "myuser"
}
```

5. Drag the OpenAI Adapter invoke connection into the integration canvas.
6. On the Basic Info page, select **Chat completions API** from the **Action** list.
7. On the Summary page, click **Finish**.
8. Open the request mapper.
9. In the request mapper, map source elements to target elements. There are differences with how the payload is handled between the Chat Completions API and the Responses API:
   - With the Chat Completions API, target subelements such as **Role** and **Content** are mapped under the **Messages** element. This differs from the Responses API, where target subelements such as **Role** and **Content** are mapped under the **Input** element.
   - The target **Tool Calls** element handles subsequent prompt calls where your response is dependent on another tool. For example, in the first prompt, you get the type of document by calling the document tool. For the second prompt, you want to create an expense report. You call the expense report tool based on the existing document tool call. An ID value is used to call the next prompt. The target **Tool Calls** element is similar to the target **Functions** element available with the Responses API.



10. In the response mapper, map source elements to target elements. There are differences with how the payload is handled between the Chat Completions API and the Responses API:
    - Under the source **Body** element are the **Choices**, **Message**, and **Tool Calls** subelements. This differs from the Responses API, where the sources **Output** and **Function** are used.



11. Complete integration design, then activate the integration.

12. Run the integration from the Configure and run page, then view your results.