Subscribe

October 13, 2024 |0 view(s) |0 people thought this was helpful

Autonomous AI Agent for performing actions

In this article

The Autonomous AI Agent for performing actions can handle various tasks, including:

- Natural Language Processing (NLP)—Understand and respond to human language in a natural and conversational manner.
- Decision making—Make informed choices based on available information and predefined rules.
- Automation—Automate repetitive or time-consuming tasks.

Create an Autonomous AI Agent for performing actions

1. Log in to the Webex AI Agent platform.
2. On the **Dashboard**, click **+Create Agent**.
3. On the **Create an AI Agent** screen, click **Start from Scratch**.

> You can also choose a pre-defined template to create your AI agent quickly. Filter the AI Agent type as Autonomous. In this case, the fields on the Profile page auto- populates.

4. Click **Next**.
5. In the **What type of agent are you building** section, click **Autonomous**.
6. In the **What's your agent's main function** section, click **Perform Actions**.
7. Click **Next**.

8. On the **Profile** page, specify the following details:
    a. **Agent name**—Enter the name of the AI Agent.
    b. **System ID**—A system-generated unique identifier. This field is editable.
    c. **AI engine**—Select the AI engine from the drop-down list.
    d. **Agent's goal**—Provide a brief description of the AI Agent's goal. See Guidelines for defining goals and instructions for AI Agent for best practices and examples.

9. Click **Create**. The Autonomous AI Agent for performing actions is successfully created and is now available on the dashboard.

> You can also import the pre-built AI agents. For more information, see Import pre-built AI Agent

What to do next

Update the profile for the Autonomous AI agent.

Update Autonomous AI Agent profile

Before you begin

Create an Autonomous AI agent for performing actions.

1. On the **Dashboard**, click the AI Agent that you've created.

2. Navigate to **Settings** > **Profile** tab and configure the following details:
    a. **Agent name**—Edit the name of the AI Agent, if needed.
    b. **System ID**—Edit the system id of the AI Agent, if needed.
    c. **Agent's goal**—Edit the goal of the AI Agent, if needed. See Guidelines for defining goals and instructions for AI Agent for guidelines and examples.
    d. **Instructions**—Enter the instructions for the AI Agent. See Guidelines for defining goals and instructions for AI Agent for guidelines and examples.
    e. **URL for Agent Logo/Brand Image**—Enter the URL from where the system fetches the AI Agent's logo or image.
    f. **AI Engine**—Update the AI engine, if needed.
    g. **Welcome message**—Enter the welcome message that the AI Agent should use to start the interaction.
3. Click **Publish** to make the AI Agent live.

What to do next

Add the required **Actions** to the AI Agent.

Add actions to Autonomous AI Agent

The Autonomous AI Agents for performing actions are designed to comprehend user utterances and act accordingly. For example, in a restaurant there is a need to automate online food order intake. To accomplish the task, you can create an Autonomous AI Agent that performs the following actions:

- Gather the required information from the customer.
- Transfer the information to the required flow.

The Autonomous AI Agent to perform actions works on the following building blocks:

- Action—A functionality that allows the AI agent to connect with an external systems to perform complex tasks.
- Entity or slot—Represents a step in fulfilling the user's intent. Slot filling involves asking specific questions to the customer to fulfill the customer's intent based on utterances. It is the trigger for an AI Agent to start performing an action. Define the input entities as part of slot filling.
- Fulfillment—Determines how the AI Agent completes the action. As part of fulfillment, define the output entities for the Autonomous AI Agent to generate the answer in a specific format. The system sends the output entities to the flow to continue with the action and complete the task successfully.

To add actions for the autonomous AI agent:

In the **Action** tab, click **New Action**. Specify the following details:

a. **Action Name**—Enter the name of the Action. Example Book tickets.

b. **Action Description**—Provide a brief description about the action.

c. **Action scope**—Select the scope of the action from the drop-down list. The following options are available:

- **Slot filling**—Define the required input entities for slot filling.
- **Slot filling and fulfillment**—Define the required input entities for slot filling and provide the details for fulfillment of the action.

Agent handover is the default action that is enabled by default. Use the toggle option to disable agent handover.

What to do next

Configure slots and define fulfillment details.

Configure slots and define fulfillment

1. In the **Slot Filling** section of the **Actions** page, add the input entities. You can add the entities one by one in table format. You can also use the JSON file and define the entities. See A Tour of JSON Schema for details.
2. To add an input entity, click **New input entity**. In the **Add a new input entity** screen, specify the following details:
   a. **Entity name**—Enter the name of the input entity.
   b. **Entity type**—Select the data type of the entity.
   c. **Entity description**—Provide a brief description about the entity.
   d. **Entity examples**—Enter an example for the entity. Click **+Add** to add more examples.
   e. Select the **Required** check box to make this entity a mandatory field.
3. Click **Add** to add the input entity. You can add as many input entities as you need.

4. Use the **Control** option to perform the following actions on the entity:
    a. Edit the entity.
    b. Delete the entity.

5. Configure the fulfillment details for implementing the AI Agent in a contact center. Specify the following details:
    a. **Select a fulfillment approach**—Select the flow.
    b. **Select a flow builder**—Select the required flow builder.
    c. **Flow webhook URL**—Enter the webhook URL. For more information, see https://developers.webexconnect.io/reference/custom-event-v2.
    d. **Services key**—Enter the service key details.

6. Configure the output entities such that the AI Agent generates the result in a format that is understandable by the flow.

7. To add an output entity, click **New output entity**. In the **Add a new output entity** screen, specify the following details:
    a. **Entity name**—Enter the name of the output entity.
    b. **Entity type**—Select the data type of the entity.
    c. **Entity description**—Provide a brief description about the entity.
    d. **Entity examples**—Enter an example for the entity. Click **+Add** to add more examples.
    e. Select the **Required** check box to make this entity a mandatory field.

8. Click **Add** to add the output entity. You can add as many output entities as you need.

9. Use the **Control** option to perform the following actions on the entity:
    a. Edit the entity.
    b. Delete the entity.

10. Click **Add** to complete the cofiguration.

What to do next

Click **Preview** to preview the AI agent. Click **Publish** to make the AI Agent live.

Add language(s) for Autonomous AI Agent

1. Go to **Settings** > **Language**
2. Select the AI Agent supported languages from the drop-down list.

What to do next

Click **Preview** to preview the AI agent. For more information, see Preview. Click **Publish** to make the AI Agent live.

After you configure the AI Agent:

- To view the AI Agent performance, see View Autonomous AI Agent Performance using analytics.
- To see the Session and History details, see View Autonomous AI Agent sessions and history.

Guidelines for defining goals and instructions for AI Agent

This article outlines the guidelines for defining goals and instructions for the Large Language Models (LLM) powered Autonomous AI Agents.

Best practices for defining goals

This section outlines best practices for writing goal prompts for the AI Agent that uses Large Language Models (LLMs) and actions to fulfill user intents. The AI Agent supports two types of actions—Slot filling and Slot filling with fulfillment. Each slot represents a step in fulfilling the user's intent.

The goal prompt should provide a clear direction for the AI Agent's purpose without delving into specifics. The details

of how to achieve the goal, including sources, destinations, dates, or other specific information, is handled by the individual actions and their slot-filling processes.

Practices to adopt

- Keep the goal general and broad.
- Focus on the overall function or purpose of the Agent.
- Use action verbs to describe the agent's primary function.
- Consider the result or benefit for the user.
- Use clear and concise language.
- Ensure that the goal aligns with the actions and capabilities of the agent.

Good Examples

- Helping users book flights
- Assisting with restaurant reservations
- Providing weather forecasts
- Managing personal task lists

Practices to avoid

- Don't include specific details like locations, dates, or user information.
- Avoid mentioning particular actions or implementation methods.
- Don't use technical jargon or complex terminology.
- Avoid overly long or complicated goal statements.
- Don't include multiple unrelated goals in a single prompt.
- Avoid using ambiguous or vague language.

Bad Examples

- Helping users book a flight from Los Angeles to San Francisco on July 15 (too specific)
- Using a flight booking API to reserve seats on an airplane (mentions implementation details)
- Booking flights, making hotel reservations, and renting cars (multiple unrelated goals)
- Using natural language processing to understand user queries about flight bookings (too technical)
- Doing stuff related to travel (too vague)

Best practices for defining instructions

You should provide LLM-powered agents with clear and actionable instructions to ensure they perform their tasks accurately and efficiently. This section outlines best practices in a Do's and Don'ts format to guide you in writing instructions for LLM-powered agents. These agents use integrated tools for tasks such as appointment booking, customer support, and other chat/voice-based interactions.

Practices to adopt

- **Be specific and clear**
    - Clearly define the task that the agent needs to perform.
    - Use straightforward and easily understandable language.
- **Provide step-by-step instructions**

    - Break down tasks into smaller, manageable steps.
    - Ensure that steps follow a logical order.
- **Include contextual awareness**
    - Provide context to help the agent understand the task.
    - Tailor instructions to enhance the user experience.
- **Specify tool utilization**
    - Clearly indicate which tools to use.
    - Provide detailed instructions on tool usage.
- **Plan for error handling**
    - Include instructions for handling common errors.
    - Provide fallback options.
- **Engage users**
    - Include instructions on user interaction.
    - Ensure the agent confirms actions and seeks feedback.
- **Ensure adaptability**
    - Allow for adjustments based on user input.
    - Encourage continuous improvement.
- **Adhere to Ethical Considerations**
    - Ensure compliance with ethical guidelines and regulations.
    - Include measures to mitigate potential biases.

Good Examples

- **Appointment booking**—Ask the user for their preferred appointment date and time. Use the calendar tool to check availability. If available, book the appointment and confirm the details with the user. If not available, suggest the next available time slot.
- **Customer support**—Greet the user and ask for their query. If the query is about a password reset, guide them through the reset process using the authentication tool. Confirm the reset and ask if they need any further assistance.

- **Feedback collection**— After completing the interaction, ask the user to rate their experience on a scale of 1–5. Log the feedback and thank the user for their input.

Practices to avoid

- **Avoid vague instructions**—Use ambiguous language.
- **Skip steps**—Assume the agent infers steps.
- **Ignore context**—Provide instructions without context.
- **Neglect tool details**—Fail to specify which tools to use.
- **Overlook error handling**—Ignore potential errors.
- **Disregard user interaction**—Skip user engagement steps.
- **Be inflexible**—Provide rigid instructions that don't allow for changes.
- **Ignore ethical guidelines**—Overlook compliance and bias mitigation.
- **Avoid using contradictory instructions**—We shouldn’t use contradictory instructions (that confuse the agent).

Bad examples

- **Appointment booking**
  - Handle customer requests (too vague).
  - Just book the appointment (without specifying how).
- **Customer support**
  - Help the user (too vague).
  - Reset the password (without confirming user identity).
- **Feedback collection**
  - Get feedback (too vague).
  - Ask if they liked it (without a structured rating system).

JSON editor for entities

In the JSON editor view, the entities should be defined in a structured JSON format. Learn more about in the tour of JSON schema.

Structure of the input parameters in JSON editor

1. **type**: Specifies the data type of the parameters object. Here, this is always "object" to denote that the parameters are structured as an object.
2. **properties**: An object where each key represents a parameter and its associated metadata.
3. **required**: An array of strings listing the names of parameters that are mandatory.

properties Object

Each key in the properties object represents an input entity/parameter and contains another object with metadata about that parameter. The metadata should always include the following keywords:

- **type**: The data type of the parameter. The allowed types are:
    - **string**: Textual data.
    - **integer**: Numeric data without decimals.
    - **number**: Numeric data that can include decimals.
    - **boolean**: True/false values.
    - **array**: A list of items, all of which are typically the same type.
    - **object**: A complex data structure with nested properties.
- **description**: The description field provides a brief explanation of what the entity represents. This helps the AI engine understand the purpose and usage of the parameter. A description that is succinct as well as consistent with the agent instructions and action description is recommended for better accuracy.

Validation is enforced by the platform for 'type' only. 'Description' is not enforced for all entities but it is highly recommended that it's added. Other useful keywords for entity metadata:

- **enum:** The enum field lists the possible values for a parameter. This is useful for parameters that should only accept a limited set of values. Developers can define custom lists of values that a parameter should accept using this.
- **pattern:** The pattern field is used with string types to specify a regular expression that the string must match. This is particularly useful for validating specific formats, such as phone numbers, postal codes, or custom identifiers.
- **examples:** The examples field provides one or more examples of valid values for the parameter. This helps the AI engine understand what kind of data is expected and can be especially useful for interpretation and validation purposes.

There are other keywords that can make the entity definition more accurate and robust, learn more about them in the aforementioned JSON schema tour.

Example

Here's a detailed example that includes various types of entities and keywords:

```
{
    "type": "object",
    "properties": {
        "username": {
            "type": "string",
            "description": "The unique username for the account.",
            "minLength": 3,
            "maxLength": 20
        },
        "password": {
            "type": "string",
            "description": "The password for the account.",
            "minLength": 8,
            "format": "password"
        },
        "email": {
            "type": "string",
            "description": "The email address for the account.",
            "pattern": "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
        },
        "birthdate": {
            "type": "string",
            "description": "The birthdate of the user.",
            "examples": ["mm/dd/YYYY"]
        },
        "preferences": {
            "type": "object",
            "description": "User preferences settings.",
            "properties": {
                "newsletter": {
                    "type": "boolean",
                    "description": "Whether the user wants to receive
newsletters.",
                    "default": true
                },
                "notifications": {
                    "type": "string",
                    "description": "Preferred notification method.",
                    "enum": ["email", "sms", "push"]
                }
            }
        },
        "roles": {
            "type": "array",
            "description": "List of roles assigned to the user.",
            "items": {
```

```
                "type": "string",
                "enum": ["user", "admin", "moderator"]
            }
        }
    },
    "required": ["username", "password", "email"]
}
```

In this example the user defined have:

- **username**: A string type with minimum and maximum length constraint.
- **password**: A string type with a minimum length and a specific format (password indicates it should be handled securely).
- **email**: A string type with a regex pattern to ensure it's a valid email address.
- **birthdate**: A string type with examples to prescribe the format of the date.
- **preferences**: An object type with nested properties (newsletter and notifications), including a boolean with a default value and a string with specific allowed values (enum).
- **roles**: An array type where each item is a string limited to specific values (enum).
- Out of all the entities, only username, password and email are mandatory as defined by the 'required' array.

All the entities in this example have descriptive names, clear descriptions and follow consistent structure and naming convention. These practices allow users to create well-defined entities that are easy for the AI engine to interpret and enforce.