

Integrate custom tools into your agent

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/1-introduction>

Introduction

Completed

- 2 minutes

Microsoft Foundry Agent Service offers a seamless way to build an agent without needing extensive AI or machine learning expertise. By using tools, you can provide your agent with functionality to execute actions on your behalf.

The AI Agent Service provides built-in tools for gathering knowledge and generating code, which provide your agent with some powerful functionality. However, sometimes your agent needs to be able to complete specific tasks or actions that an AI model would struggle to handle on its own. To accomplish these actions, you can provide your agent a *custom tool* based on your own code or a third-party service or API.

Imagine you're working in the retail industry, and your company is struggling with managing customer inquiries efficiently. The customer support team is overwhelmed with repetitive questions, leading to delays in response times and decreased customer satisfaction. By using Foundry Agent Service with custom tools, you can create a custom FAQ agent that handles common inquiries. This agent can be provided with a set of custom tools to look up customer orders, freeing up your support team to focus on more complex issues.

In this module, you'll learn how to utilize custom tools in Foundry Agent Service to enhance productivity, improve accuracy, and create tailored solutions for specific needs.

2. Why use custom tools

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/2-why-use-custom-tools>

Why use custom tools

Completed

- 3 minutes

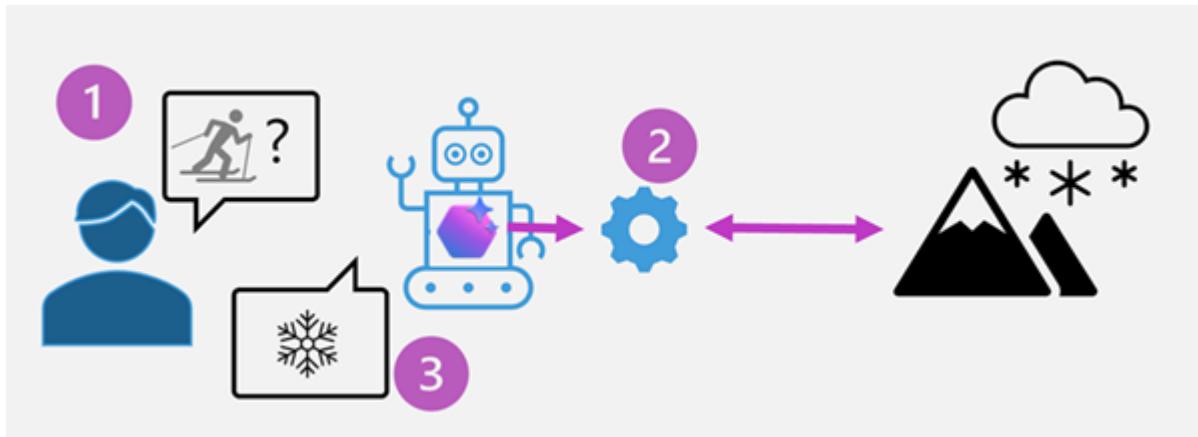
Microsoft Foundry Agent Service offers a powerful platform for integrating custom tools to enhance productivity and provide tailored solutions for specific business needs. By using these custom tools, businesses can achieve greater efficiency and effectiveness in their operations.

Why use custom tools?

Custom tools significantly enhance productivity by automating repetitive tasks and streamlining workflows that are specific to your use case. These tools improve accuracy by providing precise and consistent outputs, reducing the likelihood of human error. Additionally, custom tools offer tailored solutions that address specific business needs, enabling organizations to optimize their processes and achieve better outcomes.

- **Enhanced productivity:** Automate repetitive tasks and streamline workflows.
- **Improved accuracy:** Provide precise and consistent outputs, reducing human error.
- **Tailored solutions:** Address specific business needs and optimize processes.

Adding tools makes custom functionality available for the agent to use, depending on how it decides to respond to the user prompt. For example, consider how a custom tool to retrieve weather data from an external meteorological service could be used by an agent.



The diagram shows the process of an agent choosing to use the custom tool:

1. A user asks an agent about the weather conditions in a ski resort.
2. The agent determines that it has access to a tool that can use an API to get meteorological information, and calls it.
3. The tool returns the weather report, and the agent informs the user.

Common scenarios for custom tools in agents

Custom tools within the Foundry Agent Service enable users to extend the capabilities of AI agents, tailoring them to meet specific business needs. Some example use cases that illustrate the versatility and impact of custom tools include:

Customer support automation

- **Scenario:** A retail company integrates a custom tool that connects the Azure AI Agent to their customer relationship management (CRM) system.
- **Functionality:** The AI agent can retrieve customer order histories, process refunds, and provide real-time updates on shipping statuses.
- **Outcome:** Faster resolution of customer queries, reduced workload for support teams, and improved customer satisfaction.

Inventory management

- **Scenario:** A manufacturing company develops a custom tool to link the AI agent with their inventory management system.
- **Functionality:** The AI agent can check stock levels, predict restocking needs using historical data, and place orders with suppliers automatically.
- **Outcome:** Streamlined inventory processes and optimized supply chain operations.

Healthcare appointment scheduling

- **Scenario:** A healthcare provider integrates a custom scheduling tool with the AI agent.

- **Functionality:** The AI agent can access patient records, suggest available appointment slots, and send reminders to patients.
- **Outcome:** Reduced administrative burden, improved patient experience, and better resource utilization.

IT Helpdesk support

- **Scenario:** An IT department develops a custom tool to integrate the AI agent with their ticketing and knowledge base systems.
- **Functionality:** The AI agent can troubleshoot common technical issues, escalate complex problems, and track ticket statuses.
- **Outcome:** Faster issue resolution, reduced downtime, and improved employee productivity.

E-learning and training

- **Scenario:** An educational institution creates a custom tool to connect the AI agent with their learning management system (LMS).
- **Functionality:** The AI agent can recommend courses, track student progress, and answer questions about course content.
- **Outcome:** Enhanced learning experiences, increased student engagement, and streamlined administrative tasks.

These examples demonstrate how custom tools within the Foundry Agent Service can be used across industries to address unique challenges, drive efficiency, and deliver value.

3. Options for implementing custom tools

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/3-custom-tool-options>

Options for implementing custom tools

Completed

- 6 minutes

Microsoft Foundry Agent Service offers various custom tools that enhance the capabilities and efficiency of your AI agents. These tools allow for scalable interoperability with various applications, making it easier to integrate with existing infrastructure or web services.

Custom tool options available in Microsoft Foundry Agent Service

Foundry Agent Service provides several custom tool options, including OpenAPI specified tools, Azure Functions, and function calling. These tools enable seamless integration with external APIs, event-driven applications, and custom functions.

- **Custom function:** Function calling allows you to describe the structure of custom functions to an agent and return the functions that need to be called along with their arguments. The agent can dynamically identify appropriate functions based on their definitions. This feature is useful for integrating custom logic and workflows, in a selection of programming languages, into your AI agents.
- **Azure Functions:** Azure Functions enable you to create intelligent, event-driven applications with minimal overhead. They support triggers and bindings, which simplify how your AI Agents interact with external systems and services. Triggers determine when a function executes, while bindings facilitate streamlined connections to input or output data sources.
- **OpenAPI specification tools:** These tools allow you to connect your Azure AI Agent to an external API using an OpenAPI 3.0 specification. This provides standardized, automated, and scalable API integrations that enhance the capabilities of your agent. OpenAPI specifications describe HTTP APIs, enabling people to understand how an API works, generate client code, create tests, and apply design standards.
- **Azure Logic Apps:** This action provides low-code/no-code solutions to add workflows and connects apps, data, and services with the low-code Logic App.

This flexibility to integrate custom functionality in multiple ways enables a wide range of extensibility possibilities for your Foundry Agent Service agents.

4. How to integrate custom tools

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/4-how-use-custom-tools>

How to integrate custom tools

Completed

- 7 minutes

Custom tools in an agent can be defined in a handful of ways, depending on what works best for your scenario. You may find that your company already has Azure Functions implemented for your agent to use, or a public OpenAPI specification gives your agent the functionality you're looking for.

Function Calling

Function calling allows agents to execute predefined functions dynamically based on user input. This feature is ideal for scenarios where agents need to perform specific tasks, such as retrieving data or processing user queries, and can be done in code from within the agent. Your function may call out to other APIs to get additional information or initiate a program.

Example: Defining and using a function

Start by defining a function that the agent can call. For instance, here's a fake snowfall tracking function:

```
import json

def recent_snowfall(location: str) -> str:
    """
    Fetches recent snowfall totals for a given location.
    :param location: The city name.
    :return: Snowfall details as a JSON string.
    """
    mock_snow_data = {"Seattle": "0 inches", "Denver": "2 inches"}
    snow = mock_snow_data.get(location, "Data not available.")
    return json.dumps({"location": location, "snowfall": snow})

user_functions: Set[Callable[..., Any]] = {
    recent_snowfall,
}
```

Register the function with your agent using the Azure AI SDK:

```
# Initialize agent toolset with user functions
functions = FunctionTool(user_functions)
toolset = ToolSet()
toolset.add(functions)
agent_client.enable_auto_function_calls(toolset=toolset)

# Create your agent with the toolset
agent = agent_client.create_agent(
    model="gpt-4o-mini",
    name="snowfall-agent",
    instructions="You are a weather assistant tracking snowfall. Use the provided functions to ans
```

```
    toolset=toolset  
)
```

The agent can now call *recent_snowfall* dynamically when it determines that the prompt requires information that can be retrieved by the function.

Azure Functions

Azure Functions provide serverless computing capabilities for real-time processing. This integration is ideal for event-driven workflows, enabling agents to respond to triggers such as HTTP requests or queue messages.

Example: Using Azure Functions with a queue trigger

First, develop and deploy your Azure Function. In this example, imagine we have a function in our Azure subscription to fetch the snowfall for a given location.

When your Azure Function is in place, integrate add it to the agent definition as an Azure Function tool:

```
storage_service_endpoint = "https://<your-storage>.queue.core.windows.net"  
  
azure_function_tool = AzureFunctionTool(  
    name="get_snowfall",  
    description="Get snowfall information using Azure Function",  
    parameters={  
        "type": "object",  
        "properties": {  
            "location": {"type": "string", "description": "The location to check snowfall."},  
        },  
        "required": ["location"],  
    },  
    input_queue=AzureFunctionStorageQueue(  
        queue_name="input",  
        storage_service_endpoint=storage_service_endpoint,  
    ),  
    output_queue=AzureFunctionStorageQueue(  
        queue_name="output",  
        storage_service_endpoint=storage_service_endpoint,  
    ),  
)  
  
agent = agent_client.create_agent(  
    model=os.environ["MODEL_DEPLOYMENT_NAME"],  
    name="azure-function-agent",  
    instructions="You are a snowfall tracking agent. Use the provided Azure Function to fetch snow
```

```
    tools=azure_function_tool.definitions,  
)
```

The agent can now send requests to the Azure Function via a storage queue and process the results.

OpenAPI Specification

OpenAPI defined tools allow agents to interact with external APIs using standardized specifications. This approach simplifies API integration and ensures compatibility with various services. The Foundry Agent Service uses OpenAPI 3.0 specified tools.

Tip

Currently, three authentication types are supported with OpenAPI 3.0 tools: *anonymous*, *API key*, and *managed identity*.

Example: Using an OpenAPI specification

First, create a JSON file (in this example, called *snowfall_openapi.json*) describing the API.

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "Snowfall API",
    "version": "1.0.0"
  },
  "paths": {
    "/snow": {
      "get": {
        "summary": "Get snowfall information",
        "parameters": [
          {
            "name": "location",
            "in": "query",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Successful response",
            "content": {
              "application/json": {

```

Then, register the OpenAPI tool in the agent definition:

```
from azure.ai.agents.models import OpenApiTool, OpenApiAnonymousAuthDetails

with open("snowfall.openapi.json", "r") as f:
    openapi_spec = json.load(f)

auth = OpenApiAnonymousAuthDetails()
openapi_tool = OpenApiTool(name="snowfall_api", spec=openapi_spec, auth=auth)

agent = agent_client.create_agent(
    model="gpt-4o-mini",
    name="openapi-agent",
    instructions="You are a snowfall tracking assistant. Use the API to fetch snowfall data.",
    tools=[openapi_tool]
)
```

The agent can now use the OpenAPI tool to fetch snowfall data dynamically.

Note

One of the concepts related to agents and custom tools that developers often have difficulty with is the *declarative* nature of the solution. You don't need to write code that explicitly *calls* your custom tool functions - the agent itself decides to call tool functions based on messages in prompts. By providing the agent with functions that have meaningful names and well-documented parameters, the agent can "figure out" when and how to call the function all by itself!

By using one of the available custom tool options (or any combination of them), you can create powerful, flexible, and intelligent agents with Foundry Agent Service. These integrations enable

seamless interaction with external systems, real-time processing, and scalable workflows, making it easier to build custom solutions tailored to your needs.

5. Exercise - Build an agent with custom tools

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/5-exercise>

Exercise - Build an agent with custom tools

Completed

- 30 minutes

Now it's your opportunity to build an agent with custom tools. In this exercise, you create an agent in code and connect the tool definition to a custom tool function.

Note

If you don't have an Azure subscription, and you want to explore Microsoft Foundry, you can [sign up for an account](#), which includes credits for the first 30 days.

Launch the exercise and follow the instructions.

[Launch Exercise](#)

Tip

After completing the exercise, if you're finished exploring Azure AI Agents, delete the Azure resources that you created during the exercise.

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/6-knowledge-check>

Module assessment

Completed

- 3 minutes

7. Summary

<https://learn.microsoft.com/en-us/training/modules/build-agent-with-custom-tools/7-summary>

Summary

Completed

- 2 minutes

In this module, we covered the benefits of integrating custom tools into Foundry Agent Service to boost productivity and provide tailored business solutions. By providing custom tools to our agent, we can optimize processes to meet specific needs, resulting in better responses from your agent.

The techniques learned in this module enable businesses to generate marketing materials, improve communications, and analyze market trends more effectively, all through custom tools. The integration of various tool options in the AI Agent Service, from Azure Functions to OpenAPI specifications, allows for the creation of intelligent, event-driven applications that use well-established patterns already used in many businesses.

Further reading

- [AI Agents for beginners tool use](#)
- [Microsoft Foundry Agent Service function calling](#)
- [Introduction to Azure Functions](#)
- [OpenAPI Specification](#)

