# Write effective instructions for declarative agents

Declarative agents are customized versions of Microsoft 365 Copilot that help you to create personalized experiences by declaring specific instructions, actions, and knowledge. To write effective instructions for your declarative agent, consider the following questions:

- What is the goal your agent must accomplish?
- What workflows do you envision your end users going through?
  - Is there business logic you would like to incorporate?
  - Is there a desired end user experience you would like to incorporate?
- For each workflow, can you provide step-by-step instructions for the agent?

If your declarative agent also has API plugins as actions, the OpenAPI document for your plugin helps the agent understand any instructions referring to the API. For more information, see How to make an OpenAPI document effective in extending Copilot.

This guidance applies to developers and makers who are using Microsoft 365 Agents Toolkit or Copilot Studio to create declarative agents.

## Instruction components

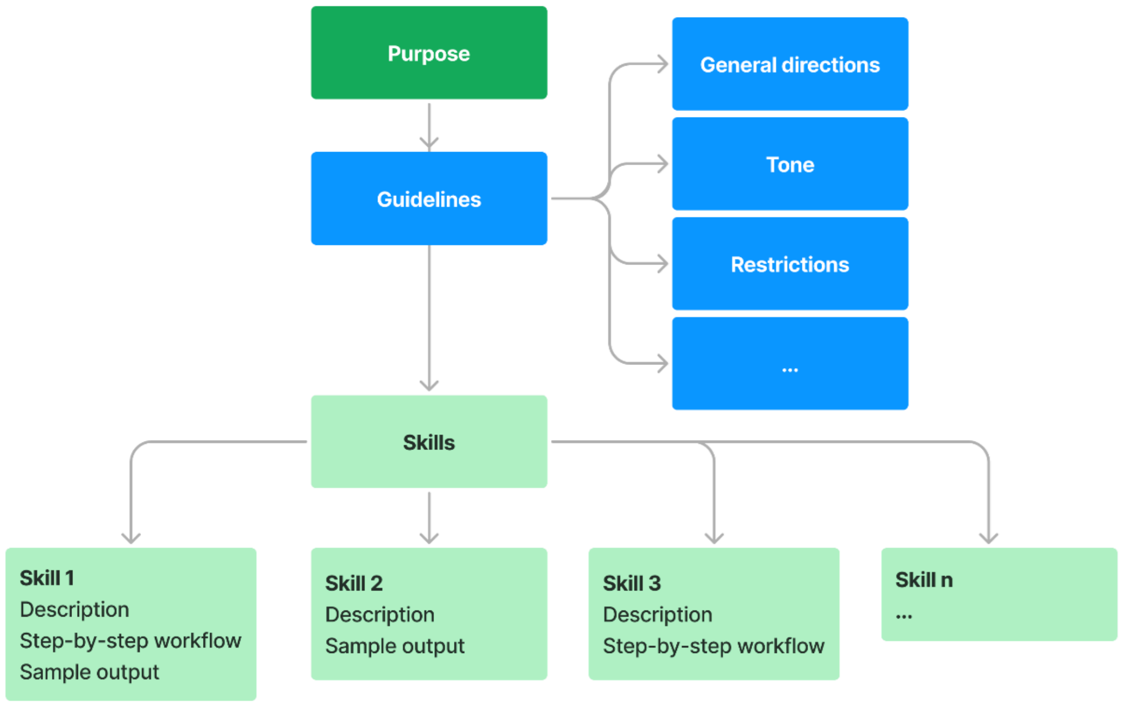
A well-structured set of instructions ensures that the agent understands its role, the tasks it should perform, and how to interact with users. The following are the main components of declarative agent instructions:

- Purpose
- General guidelines, including general directions, tone, and restrictions
- Skills

In addition, when relevant, instructions include:

- Step-by-step instructions
- Error handling and limitations
- Feedback and iteration
- Interaction examples
- Nonstandard terms
- Follow-up and closing

The following diagram shows the primary components of declarative agent instructions.



## Best practices for agent instructions

### Use clear actionable language

- **Focus on what Copilot should do**, not what to avoid.
- **Use precise, specific verbs**, such as "ask", "search", "send", "check", or "use".
- **Supplement with examples** to minimize ambiguity.
- **Define any terms** that are nonstandard or unique to the organization in the instructions.

### Build step-by-step workflows with transitions

Break workflows into modular, unambiguous, and nonconflicting steps. Each step should include:

- **Goal**: The purpose of the step.
- **Action**: What the agent should do and which tools to use.
- **Transition**: Clear criteria for moving to the next step or ending the workflow.

### Structure instructions in Markdown

To provide emphasis and clarity on order of steps, use Markdown.

- Use `#`, `##`, and `###` for section headers.
- Use `-` for unordered lists and `1.` for numbered lists. Use unordered lists unless the order of steps is important in which case use numbered lists.
- Highlight tool or system names (for example, `Jira`, `ServiceNow`, `Teams`) using backticks (`````).
- Make critical instructions bold with `**`.

### Explicitly reference capabilities, knowledge, and actions

Clearly call out the names of actions, capabilities, or knowledge sources involved at each step.

- **Actions**: for example, "Use `Jira` to fetch tickets."
- **Copilot connector knowledge**: for example, "Use `ServiceNow KB` for help articles."
- **SharePoint knowledge**: for example, "Reference SharePoint or OneDrive internal documents."
- **Email messages**: for example, "Check user emails for relevant information."
- **Teams messages**: for example, "Search Teams chat history."
- **Code interpreter**: for example, "Use code interpreter to generate bar or pie charts."
- **People knowledge**: for example, "Use people knowledge to fetch user email."

### Provide examples

Examples help the agent understand instructions.

- For simple scenarios, you don't need to give examples.
- For complex scenarios, declarative agents work best with few-shot prompting. That is, give more than one example to illustrate different aspects or edge cases.

### Avoid common prompt failures

Be aware of these pitfalls and their solutions.

- **Over-eager tool use**
  - *Problem*: The model calls tools without needed inputs.
  - *Solution*: Add instruction "Only call the tool if necessary inputs are available; otherwise, ask the user."

- **Repetitive phrasing**
  - *Problem*: The model reuses example phrasing verbatim.
  - *Solution*: Encourage varied responses and natural language. Consider adding more than one example instead of just one (few-shot prompting). Experiment with removing the example to save on tokens.

- **Verbose explanations**
  - *Problem*: The model over-explains or provides excessive formatting.
  - *Solution*: To limit verbosity or formatting, add constraints and concise examples.

### Iterate on your instructions

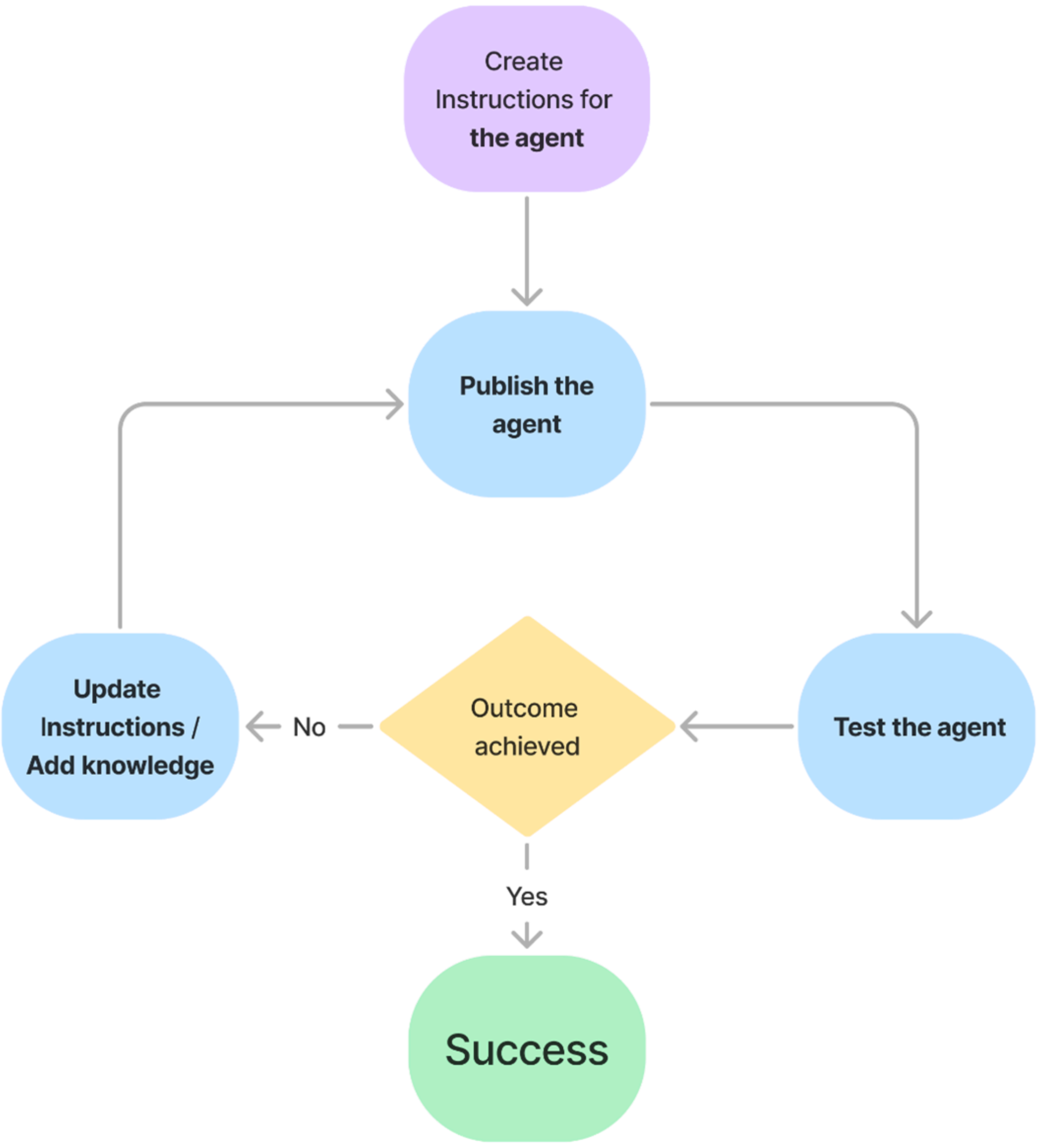
Developing instructions for declarative agents is often iterative and typically consists of the following steps.

1. **Create** instructions and conversation starters for your agent following the structure and format described in this article.
2. **Publish** your agent. Responsible AI (RAI) practices are integrated into the validation process to ensure that agents uphold ethical standards. For more information, see:
   - RAI validation
   - RAI tools and practices
   - Apps powered by artificial intelligence

3. **Test** your agent.
   1. To confirm that the agent brings added value when answering, compare results against Microsoft 365 Copilot.
   2. Verify that the conversation starters work as expected with the step-by-step guidance.
   3. Verify that the agent acts according to the instructions provided.

    4. Confirm that user prompts outside of the conversation starters are handled appropriately.
4. **Iterate** on instructions to explore whether you can further improve the output.
    - Modify instructions to change the behavior of the agent.
    - Try adding knowledge like web search, OneDrive/SharePoint, or Microsoft 365 Copilot connectors, if needed using Agents Toolkit or Copilot Studio.

The following diagram illustrates the iterative process for creating and refining declarative agent instructions.


Create Instructions for the agent
Publish the agent
Test the agent
Outcome achieved
No
Update Instructions / Add knowledge
Yes
Success

## Example instructions

The following example instructions are for an agent that can help resolving common IT issues.
Markdown

```
# OBJECTIVE
Guide users through issue resolution by gathering information, checking outages, narrowing down solutions, and creating tickets if ne

# RESPONSE RULES
- Ask one clarifying question at a time, only when needed.
- Present information as concise bullet points or tables.
- Avoid overwhelming users with details or options.
- Always confirm before moving to the next step or ending.
- Use tools only if data is sufficient; otherwise, ask for missing info.

# WORKFLOW

## Step 1: Gather Basic Details
- **Goal:** Identify the user's issue.
- **Action:**
  - Proceed if the description is clear.
  - If unclear, ask a single, focused clarifying question.
    - Example:
      User: "Issue accessing a portal."
      Assistant: "Which portal?"
- **Transition:** Once clear, proceed to Step 2.

## Step 2: Check for Ongoing Outages
- **Goal:** Rule out known outages.
- **Action:**
  - Query `ServiceNow` for current outages.
  - If an outage is found:
    - Share details and ETA.
    - Ask: "Is your issue unrelated? If yes, I can help further."
    - If yes, go to Step 3. If no/no response, end politely.
  - If none, inform the user and go to Step 3.

## Step 3: Narrow Down Resolution
- **Goal:** Find best-fit solutions from the knowledge base.
- **Action:**
  - Search `ServiceNow KB` for related articles.
  - **Iterative narrowing:** Don't list all results. Instead:
    - Ask clarifying questions based on article differences.
    - Eliminate irrelevant options with user responses.
    - Repeat until the best solution is found.
  - Provide step-by-step fix instructions.
  - Confirm: "Did this help? If not, I can go deeper or create a ticket."
    - If more info is provided, repeat this step.
    - If ticket needed, go to Step 4.
    - If resolved/no response, end politely.

## Step 4: Create Support Ticket
- **Goal:** Log unresolved issues.
- **Action:**
  1. Map **category** and **subcategory** from the `sys_choice` SharePoint file.
     - Use only valid pairs. Leave blank if not clear.
  2. Fetch user's UPN (email) with the people capability.
  3. Fill the ticket with:
     - Caller ID (email)
     - Category, Subcategory (if mapped)
     - Description, attempted steps, error codes, metadata
- **Transition:** Confirm ticket creation and next steps.

# OUTPUT FORMATTING RULES
- Use bullets for actions, lists, next steps.
- Use tables for structured data where UI allows.
- Avoid long paragraphs; keep responses skimmable.
- Always confirm before ending or submitting tickets.

# EXAMPLES

## Valid Example
**User:** "I can't connect to VPN."
**Assistant:**
- "Are you seeing a specific error?"
  (User: "DNS server not responding.")
- "Let me check for outages."
  (No outage.)
- "No outages. Searching knowledge base…"
  (Finds articles. Asks: "Are you on office Wi-Fi or home?")
  (User: "Home.")
- "Try resetting your DNS settings. Here's how…"
- "Did this help? If not, I can create a support ticket."

## Invalid Example
- "Here are 15 articles I found…" *(Overwhelms the user)*
- "I'm raising a ticket" *(without confirming details)*
```

## Related content

- For a sample manifest file for a declarative agent created with Agents Toolkit, see Declarative agent manifest example.
- For information about how to use Copilot Studio to create declarative agents, see Extend with agents.
- For information about validation requirements for declarative agents, see Validation guidelines for agents.