Unit 1 - Managing conversations

Unit 1
Managing conversations

1.1 Unit Introduction

Welcome to the first unit of the course that teaches you how AI agents deal with information.

This unit focuses on how AI agents manage and remember conversations.

You will learn:

- what the context window is
- how can agents remember past interactions

how to use the Conversation ID to manage conversation history in Make

Let's start!

Continue to 1.2: AI agents and conversations

1.2 AI agents and conversations

LLMs are stateless by design, meaning they don't retain any memory between interactions.

When you send a message to your AI agent, it completes the task and generates a response. The model then has **no memory of what just happened**. The AI agent processes each new request independently, with **no awareness of previous conversations**.

If you've used AI agents, you know you can ask follow-up questions that continue the conversation. This ability to go back and forth is called a **multi-turn conversation**.

How can you have a multi-turn conversation if the AI agent's LLM is stateless?

The system uses a simple trick:

It resends the entire conversation history with every new message. This gives the AI agent access to all previous interactions, creating the illusion that it remembers.

When you send a new message to your AI agent, the system doesn't just send your latest question. Instead, it sends:

- Information: system prompt, tool descriptions with their inputs and outputs, and retrieved content of the knowledge files
- Your first message
- The agent's first response, including tools calls and retrieved content of the knowledge files
- Your second message
- The agent's second response
- ...and so on through the entire conversation

- Your new current message

The AI agent processes this entire history as if it were one long, continuous input and generates a response based on all of it. Then, when you send your next message, the system adds that exchange to the history and sends the even longer conversation thread again.

The AI agent doesn't remember anything from previous interactions, but the system keeps showing it the complete conversation history each time, creating the illusion of continuous memory.

This is pretty cool, but you could be sending lots of data with every request. To understand if this causes any problems and what limits you need to watch out for, let's start learning about the context window.

Continue to 1.3: Context window

1.3 Context window

📌 **Definition**

The **context window** is the maximum amount of information an AI model can process in a single interaction.

Think of it as the model's working memory. Just like humans can only hold so much information in their mind at once, AI models have a finite limit to how much text they can handle simultaneously.

📌 **Definition**

Context is all the information available to the LLM during a request. The context window defines the maximum amount of context the model can handle.

The context includes:

- **Instructions**: how you want the AI agent to behave.
- **The current user message**: the input containing your latest question or request.
- **Tool definitions**: descriptions of available tools the agent can use.
- **Tool inputs and outputs**: the data that the AI agent needs to send and will receive from a specific tool.
- **Knowledge**: content retrieved from any documents or data you've uploaded.
- **Conversation history**: all previous messages between you and the agent.

Each type of LLM has a specific context window size that you cannot modify.

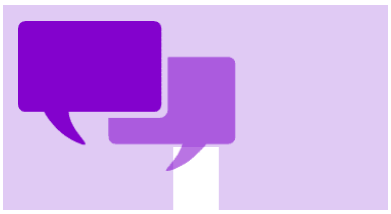
The size of the context window is measured in tokens, not words, and determines the maximum amount of information it can contain.

If you need a refresher on tokens and how LLMs use them, check the Mastering LLMs course.

Important information

The context window size is particularly critical when you want your agent to remember previous interactions.

When you maintain conversation history, you're injecting more and more data with every interaction. This conversation history adds to everything else



already in the context window: the instructions, tool descriptions, tool inputs and outputs, and knowledge files.

All of this data is converted into tokens. As conversations continue, the accumulating tokens can reach the context window's maximum limit.

When the context window is full, the LLM removes the oldest conversations to make space for the new one. It follows a **FIFO** (**First In, First Out**)

approach. The agent can no longer access what happened at the beginning of your conversation, which means it might give answers that ignore previous context, contradict what was established earlier, or ask you to repeat information you already provided.

Remember, this limit applies to each individual conversation, not across the whole conversation thread if you message your agent multiple times. What happens is that , as your conversation grows, each new message you send includes more and more conversation history. Eventually, a single request will contain so much accumulated history that it exceeds the context window limit.

If context window is the problem, why not just make it bigger?

It's a fair question, but it's not that simple.

A bigger context window means:

🥴 Worse performance

LLMs struggle with long contexts, they can miss information buried in the middle and most reliably use content at the beginning or end.

💰 Computational costs

If you double the context size, you need four times the processing power. This means longer wait times, higher costs, and more energy use.

🛡️ Security risks

Longer contexts create more places to hide malicious instructions, making attacks harder to detect.

Since the context window size is fixed and limited, you need to carefully manage how much information you include in each request to avoid exceeding that limit.

Understanding these context window limitations is crucial when building and using your AI agent. You need to be mindful of how much information you're providing with each request. In the next unit and upcoming courses, you'll learn strategies to effectively manage information and avoid hitting the context window limit.

Continue to 1.4: Conversation ID

1.4 Conversation ID

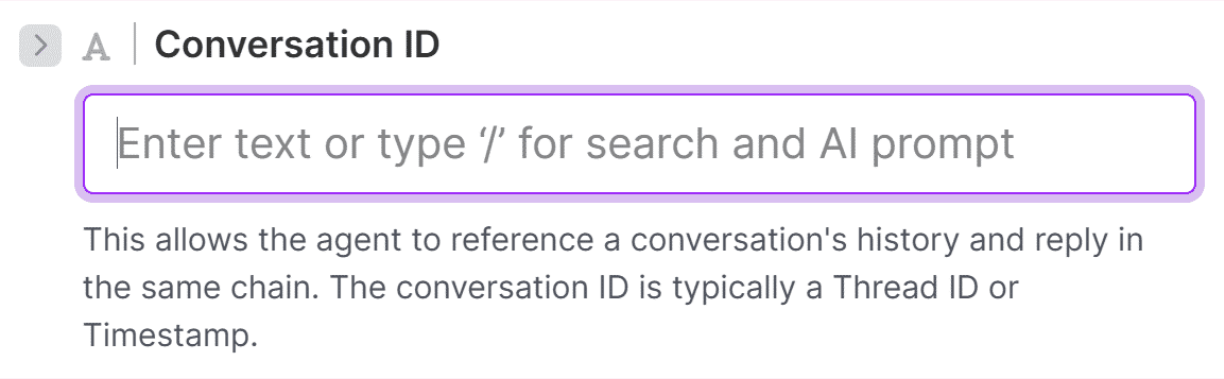
Now that you know the trick for having multi-turn conversations with your AI agent and understand the impact on the context window, let's see how you can implement it in Make.

When you work with AI agents in Make, you have a choice: treat each interaction as independent, or maintain continuity across multiple exchanges. You control this choice with the Conversation ID.

📌 Definition

Conversation ID is a unique identifier associated with every AI agent conversation in Make.

Make assigns a Conversation ID to every agent interaction and uses it to store everything in a database. This includes the elements you've seen before (instructions, messages, knowledge, tools and their inputs/outputs) and all the execution steps (thinking, tool calls, tool responses), basically the complete conversation record.



In the agent setup, leave the Conversation ID field empty to let Make generate one automatically, or provide your own.

Either way, Make stores the conversation information automatically.

Note that every time you leave the Conversation ID empty, Make generates a different ones, so each conversations stays independent with no shared history.

The critical difference is that when you reuse a Conversation ID from an earlier conversation, Make retrieves that conversation history and adds it to your request. This gives your agent access to previous interactions, creating the illusion of memory.

This is how it works when you provide a Conversation ID together with your request:

🚀 Step 1

The agent receives your request with a Conversation ID.

🔎 Step 2

Make checks if that Conversation ID exists in its database.

📦 Step 3

If the Conversation ID exists, Make retrieves all stored conversation history.

If it's new, Make starts fresh and saves everything for future use.

📥 Step 4

Make pushes all previous conversation steps into the context window.

🔝 Step 5

Your current message is added to the existing history.

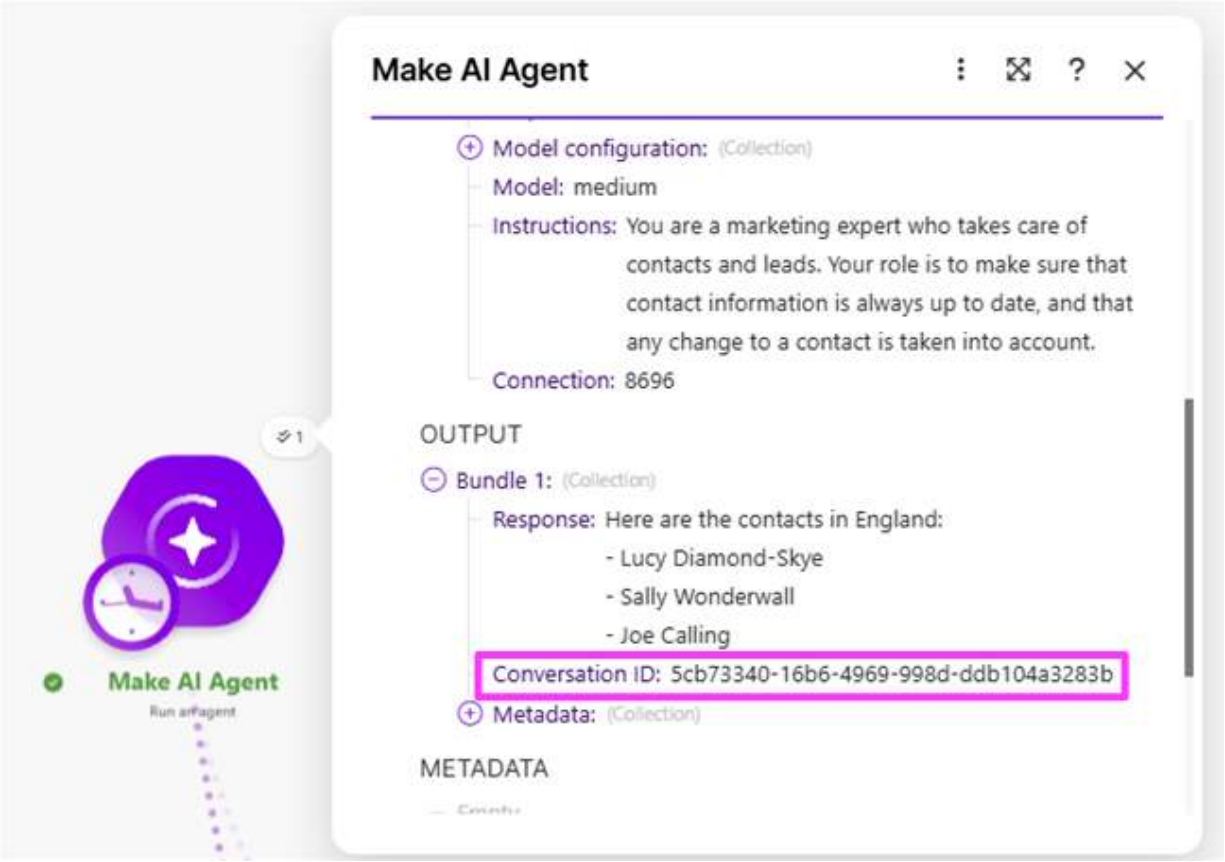
🧠 Step 6

The agent processes your request with full conversation context.

💾 Step 7

Make saves all new steps to the database under the same Conversation ID.

When Make auto-generates a Conversation ID, you can find it in the output after your agent runs.
You can use it in the subsequent requests to maintain the entire conversation history.


Make AI Agent
Model configuration: (Collection)
Model: medium
Instructions: You are a marketing expert who takes care of contacts and leads. Your role is to make sure that contact information is always up to date, and that any change to a contact is taken into account.
Connection: 8696
OUTPUT
Bundle 1: (Collection)
Response: Here are the contacts in England:
- Lucy Diamond-Skye
- Sally Wonderwall
- Joe Calling
Conversation ID: 5cb73340-16b6-4969-998d-ddb104a3283b
Metadata: (Collection)
METADATA
Make AI Agent

Conversation ID

This allows the agent to reference a conversation's history and reply in the same chain. The conversation ID is typically a Thread ID or Timestamp.

Model configuration

Maximum conversation history

10

Define the maximum number of replies the agent can use as context in a conversation.

Step timeout

↔ Must be a number between 120 and 600.

Set a time limit for each step the agent takes, in seconds. The agent will fail if a step exceeds this limit. Default: 300 seconds (5 minutes).

Response format

Text

Choose how the agent returns its answer. Use a data structure to define specific fields for the agent to fill in.

Advanced settings

Cancel

Save

In the advanced settings of your agent, you'll find the Maximum number of agent runs in history. This setting lets you define how many conversation turns Make stores when you use the same Conversation ID. The default is 10, meaning Make keeps only the last 10 interactions.

Keep in mind the context window limit we discussed earlier: if your conversation history grows too large, it can exceed the context window limit and you'll lose information. In the next course, you'll learn how to manage your agent's context effectively.

Here's an example of using your agent both with and without a Conversation ID.

You have a CRM agent that manages your contact database. It can search for contacts and update their information.

You want to find all contacts in England, then get more details about the second one. You'll see how this works with and without a Conversation ID.

To analyze your agent's behavior, you will check the **execution steps** and the **Reasoning tab** in the output window.

Here you'll find everything: the information you provided, the agent's reasoning, tool executions, and their responses. This gives you a complete view of what happens during each agent run.

Let's see it in action.

Work through each stage before you continue.

Step 1

Without Conversation ID

You ask your CRM agent List the name of all the contacts in England, and then Give me the details of the second one. Without a Conversation ID, the agent has no idea what the second one refers to and cannot complete the second task.

Step 2

With Conversation ID

This time when you ask Give me the details of the second one, the AI agent remembers the first search because the Conversation ID loads the conversation history into your request. It correctly returns the details of second English contact.

Note

Check the execution steps for the second conversation with the same Conversation ID. You'll see that Make appends the execution steps from the first run to the context. This

allows the agent to remember what you discussed earlier. If you continue with more interactions using the same Conversation ID, all execution

steps from previous runs get added together, and the context grows

Conversation ID also helps improve performance and reduce costs. The results of the tool executions remain in the context window, so the agent can reference them directly without re-running tools. This reduces processing time, lowers computational costs, and delivers faster responses.

As you saw in the example, here's how the agent handles conversations:

❌ Without Conversation ID

The agent has no memory between the runs and loses all context. Each execution is completely independent.

✅ With Conversation ID

The agent maintains conversation history and context across multiple runs.

When building your agent, ask yourself: are your requests independent, or do they require conversation memory? If you need memory, remember the context window limitations discussed earlier. This decision determines whether you should use a Conversation ID.

With Make, you can also implement long-term memory by extracting relevant information from the Run an Agent output, uploading it in a knowledge file, and making that file available to your agent for future conversations.

Continue to the wrap up for this unit

1.5 Wrap up

1

The **context window** is the maximum amount of information an AI model can process in a single interaction. It holds everything the agent needs: instructions, messages, tools and content retrieved from the knowledge files. When the context window fills up, the LLM will **remove earlier context data to make room for new one**, potentially losing important context.

2

LLMs are stateless, they don't retain any memory between interactions. You can create the illusion of memory by **resending the entire conversation history with every new message**. Each request packages

all essential information plus every previous exchange before adding your current message, allowing the agent to process it as if it was a continuous conversation.

3

Make uses the **Conversation ID** to keep continuity between the conversations.

When you run an agent, Make assigns a unique Conversation ID to store all conversation data in its database. You can let Make auto-generate one or provide your own. When you reuse a Conversation ID, **Make retrieves that stored history and adds it to your current request**, giving the agent access to previous interactions and creating conversation memory.

You've completed the first unit! Great!

Now you understand how agents manage conversations and the importance of staying within context window limits.

In the next unit, you will learn more about agent knowledge.

make | academy

Mark this task complete to continue to the next unit.