Tab 1

portalis USER GUIDE

MILCODE USAGE DOCUMENTATION

Types of Documents

Semantic Searches (Doc and Memory)

Semantic documents are specially formatted data files that the Avatar can use to pull in relevant information during a conversation. When structured correctly, these documents act like a real-time reference — almost like a mini database the Avatar can search through as it responds to the user. This is done using the appropriate MILCode commands in your prompt.

You MUST use a simple text document to create your semantic document (.txt). After it is uploaded, the Portalis system will convert it from Filename.txt to Filename.sem. As you update the .txt file, don't forget to upload the changes as you go!

.txt File format:
—----
Subject / Title #1
Information Block #1

Subject / Title #2
Information Blob #2 (the more specific to one subject the better)
—---

Basic syntax for a Semantic Search command:
${SEMANTIC|FILENAME|3|<combo>}$

- This command will search the Filename.sem file for an average of 3 results using a combination of the last user response and the last Avatar response.

You can also replace "<combo>" with:
{{MIL_LAST_USER_RESPONSE}} or
{{MIL_LAST_SYSTEM_RESPONSE}} or
<ACTUAL TEXT OF SEARCH TERMS>

These options are useful for feeding the Avatar ongoing information as the conversation develops, or searching specific text within the document.

In addition to searching files for relevant information, you can also search the Avatar's memory of past conversations with the User. This is useful if you'd like to reference past conversation history in real time:
${REMEMBER|SEARCH TERMS|3}$

Examples:

${SEMANTIC|SearchDoc|5|<combo>}$

- Will search the file SearchDoc.sem using the text from the last Avatar response text and the last User response text

${SEMANTIC|myDoc|3|biography information}$

- Will search the file, myDoc.sem, for the phrase "biography information" to return 3 relevant results

${REMEMBER|my favorite movies|2}$

- Will search for all past conversation history for "my favorite movies" references to return the top 2 results from past interactions between the User and Avatar.

Table Doc Lookup

You can create your Table Document within the Portalis Platform. Simple click "Create File" button and follow the format:

- **Key:** This the single word or short phrase that must be used to trigger a lookup of the Table record.
- **Alias:** This is a series of words or phrases that the Avatar can use to narrow in on the correct Key to use. It can use context clues to help figure out the right Key in the lookup table.
- **Value:** This is the block of information that will be retrieved and pulled into the Avatar's brain when it completes the look up request. This could be a block of data, a step in a decision tree, location specific information, etc.
- **Lock Icon:** If Lock is turned on, then the Avatar will NOT be able to access the table's information (even with the correct key). Use this only if you are worried the Avatar will access the information when you don't want it to. Useful for things like Quizzes or private information.

Basic Syntax for a Table Lookup Command:
${LOOKUP|FILENAME|KEY}$

- This command will search the Filename.tab file for the entry with Key "KEY" and pull the Value section of that Table Entry.

Often you will embed an Image request in the Value section of a Table entry and a Lookup command will automatically pull up those images. You can pull the text information without the related images by attaching the "skip_images" parameter like this:
${LOOKUP|FILENAME|KEY|skip_images}$

You'll often want your Avatar to know what all the possible Key values are to choose from when doing lookups. You can give the Avatar a list of all the options with the LIST command:
${LIST|FILENAME|all}$

- In addition to using “all” you can also use “locked” or “unlocked” if you only want the Keys listed from those subsets.
- This list will output all of the Keys,the Lock State (locked or unlocked), and the Aliases.

Table docs can be used for some advanced strategies like generating decision trees or swapping out large prompting chunks. In these instances it can be helpful to have a starting point with a table doc. For that, we use the following Commands and Variables:
${STATEDOC_INIT|FILENAME}$
{{LASTLOOKUP_FILENAME}}

The “Statedoc_Init” Command will set the “LastLookup_FILENAME” variable the keyword in the first entry in the Table document. The great thing about this Command is that if the LastLookup_FILENAME variable is already set then the command will do nothing. It will only set the variable if its the First Time (this means you can put this command in the prompt file directly without having to worry that the variable will get reset every time the user says something).

You can then use that variable to do a Table look up as a starting point. For example:
${LOOKUP|FILENAME|{{LASTLOOKUP_FILENAME}}}$

To Lock and Unlock entries in your Table Doc (remember this is only to be used if you want to hide things from the Avatar):
${LOCK|FILENAME|KEY}$
${UNLOCK|FILENAME|KEY}$
${CHECK_LOCK|FILENAME|KEY}$ (checks to see if the KEY entry is Locked and then returns True or False)

Inventory Storage

Inventory documents are used to store lists, or containers of information outside of the Avatars memory. This is useful for tracking changing inventories or updating lists of items; or if you’re building a game, chests of magical items!

You can create your Inventory Document within the Portalis Platform. Simple click “Create File” button and follow the format:

- **Container:** First you will build your containers with Add button
- **Items:** Then you will add starter items to your containers
- The containers must all be build in the File Creation process, but ultimately the items will change as you give the Avatar command. The items you enter during the creation process will be the starter items in each container when the Avatar talks to a new person.

If your Avatar has Memory turned on, then the item changes will persist between conversations.

There are three basic Commands for Inventory docs; adding new items to a container; deleting new items; listing the containers and items. Here's an example of adding items to a birthday list set:
${INV_ADD|FILENAME|Daughter's List|Movie Poster}$
- Adds Movie Poster to Daughter's List

${INV_DEL|FILENAME|Daughter's List|3}$
- Deletes item 3 from Daugher's list

${INV_LIST|FILENAME|Daughter's List}$
- Lists all of the items in the Daughter's List

For Inventory changes, this is a good place to use the Tool Call version of the commands so the Avatar can more easily make quick changes as you talk. As a reminder, this syntax looks a little different, and you have to add a paragraph to the prompt explaining when and how you'd like the Avatar to use this Command:
[[INV_ADD|FILENAME|CONTAINER NAME|NEWITEM]]

We'd suggest using the ${INV_LIST|FILENAME|CONTAINER NAME}$ to force a constant list refresh and then use the Tool Call version for the Adding and Deleting elements. This will ensure the Avatar always knows the most up to date item numbers and container information while having full flexibility over changing the containers.

Pulling & Displaying Images, Files & Data

Displaying File Content

One of the most useful and casual MILcode Commands is LOAD_FILE. This simply inserts the contents of the named .txt file into the location of the command (often the prompt). This is useful for keeping the prompt clean, or giving the Avatar access to additional information on demand.

${LOAD_FILE|MyListOfStuff}$
- Will return the contents of the file "MyListOfStuff.txt"

Popping up Images

Your Avatar can pop up images at any time, creating a great picture-in-picture conversation to help guide users along.

I will highlight both the Forced pop up and the Tool Call version here to show how each can be used by the Avatar:

Forced Image Pop Up:
${IMAGE|<URL>}$

Tool Call Image Pop Up:
[$IMG:<URL>]
or
[[IMAGE|<URL>]]

The Forced version can be inserted into a Semantic doc, Table doc, or Inventory Container and will appear as soon as the Avatar accesses the appropriate location in the file.

The Tool Call version will be "discretionary" and used by the Avatar per the instructions you give it. For example, you might include:

- "You have a tool that allows you to display images. The format of the tool is [[IMAGE|<URL>]], where <URL> is the path to the image you want to display."

Popping up URL Links

Your Avatar can pop up URLs at any time. These links will show up at the bottom of the Avatar image and also inside of the Chat Log. If the user clicks on them, a new browser tab will open to the linked web page.

I will highlight both the Forced pop up and the Tool Call version here to show how each can be used by the Avatar:

Forced Image Pop Up:
${HREF|<URL>|Display Text}$

Tool Call Image Pop Up:
[$HREF:<URL>|Display Text]

If you don't include Display Text then the Display Text will be the URL itself.

API Documentation

Please see our API Documentation

Variables

Variables are referenced using double curly brackets. Variables are simply replaced in the prompt with their value anytime they occur.

MILCode variables are case insensitive!

Setting Variables
${SET|TestVariable|It worked!}$
${START_VALUE|OtherVariable|My first setting}$

- Start_Value will only set the variable to “My first setting” if the variable has never been used or set before.

Using Variables
${GET|TestVariable}$

- Will return “It worked!”

You can also short cut the GET command by using double curly brackets.
Writing {{TestVariable}} is the same as using ${GET|TestVariable}$

String literal tags

When text is enclosed in the string literal markers, @{...}@, the text will NOT be processed for MilCode and will instead be handled as a raw string.
e.g. This text will remain untouched allowing for json like: {{"field":"value"}}
This string would normally be interpreted as a Milcode variable and would generate an error.

Example:

${web_post|https://myurl.com/test|some_json=@{{{"field":"value"}}}@}$

Reserved Variables
Reserved variables are pre-defined in the system. (use: {{ ... }} to access them)

User / Avatar Information

- MIL_AI_NAME - The AI’s name
- MIL_AI_GENDER- The AI’s gender
- MIL_USER_NAME - The user’s name
- MIL_USER_GENDER- The user’s gender

User / AI Pronouns

- MIL_AI_GEN_SP/MIL_USER_GEN_SP - Subject Pronoun - he/she/they
- MIL_AI_GEN_OP/MIL_USER_GEN_OP - Object Pronoun - him/her/them
- MIL_AI_GEN_PA/MIL_USER_GEN_PA - Possessive Adjectives - his/her/their
- MIL_AI_GEN_PP/MIL_USER_GEN_PP - Possessive Pronoun - his/hers/theirs
- MIL_AI_GEN_RP/MIL_USER_GEN_RP - Reflexive Pronoun - himself/herself/themself

Conversation Variables

- MIL_LAST_USER_RESPONSE- Last sentence spoken by user.
- MIL_LAST_SYSTEM_RESPONSE- Last sentence spoken by the AI
- MIL_CHAT_LOG - This reserved variable has the accumulated conversation in the current session.
- MIL_CONVERSATION_ID - This reserved variable has the Portalis identifier for the current conversation.

Session Variables

- MIL_CURRENT_TURN - The current chat turn count
- MIL_TURNS_REMAINING - How many turns are left until the chat ends (if configured to have limited turns)
- MIL_SESSION_ID - An ID for this chat session

Configuration Settings

- MIL_LANGUAGE - Current selected language
- MIL_SPEED - This variable controls the speed of Avatar valid values are 0.8 to 1.2 (0.8 is slower and 1.2 is faster) Example: speech.${SET|MIL_SPEED|1.2}$
- MIL_CHAT_ARG_## - This will return the specified argument passed into the chat (example: /chat Test Red Green Blue). NOTE: The argument numbering starts at 0. So, in our example,{{MIL_CHAT_ARG_0}} is Red.

Conditionals

If Statements

${IF|TRUEFALSE_CONDITION|RESULT_IF_TRUE|RESULT_IF_FALSE}$

Example:
Secret Code Test: ${IF|{{know_secret}}|You know a secret code, it's the word MONKEY.|You don't know the secret code yet.}$

- If the variable {{know_secret}} is True this will insert “You know...”; otherwise, it will insert “You don’t know...”
- If TRUEFALSE_CONDITION is the string “True”, “Yes”, or “1” (case insensitive), then the RESULT_IF_TRUE is returned, if not then RESULT_IF_FALSE is returned.

Web Commands

Web commands are tools that enable your system to communicate with websites and online services automatically. They allow you to send data to external systems, retrieve information from web APIs, integrate with third-party services, and create automated workflows that connect your application with other online platforms.

WEB_GET

This command fetches information from websites and sends it to your avatar. It's like asking a website for data, similar to visiting a webpage in your browser. You provide the website address and any additional parameters. The command returns the text content from the webpage unless you specify 'silent' to suppress output.

Usage: ${WEB_GET|URL|parameters|silent}

Example: ${WEB_GET|https://api.weather.com|city=Chicago,temp=fahrenheit}$ would get weather data for Chicago.

WEB_POST

This command sends information from the avatar to a website, like filling out and submitting an online form. You can send multiple variables, and you can use stored values. Note that MILcode variables can be used in these values.

Usage: ${WEB_POST|website address|field1=value1|field2=value2|...}$

Example: ${web_post|https://myurl.com/test|customer_id={{cust_id}}|order=${as_json|inventory|OrderInfo}$}$

WEB_SEND

This command sends JSON contents of stored data files directly to websites without additional formatting. It takes inventory or lookup table data and transmits it as-is. You specify the URL, file type (inventory or table), and filename.

Usage: ${web_send|URL,filetype,filename}$

Example: ${web_send|https://partner.com/data,inventory,ProductList}$ sends your complete product inventory.

WEB_POST_RAW

This command sends custom raw payloads to endpoints exactly as written. It allows complete control over the message format without automatic processing. The content between @{ and }@ is sent exactly as specified.

Usage: ${web_post_raw|URL|@{custom content}@}$

Example: ${web_post_raw|https://reports.com|@{Sales Report: Today's total $1,250}@}$ sends a custom formatted message.

Misc. Commands

Date & Time:

Examples:

The current date and time: ${DATETIME}$

- This will return: "The current date and time: Thursday, October 19, 2023 01:36 PM

The current date: ${DATE}$

- This will return: The current date: Thursday, October 19, 2023

Calculations:

${CALC|<equation>}$

Example combining variable setting and math calculations:

${SET|var1|8}$
${CALC|(2+{{var1}})*3}$

- Returns 30: (8+2)*3

Contents of the expression may include the following:

- Arithmetic Expressions:
  - Addition (+), subtraction (-), multiplication (*), division (/), and modulus (%),
  - Exponents (**) and square root.
- Mathematical Functions:
  - Trigonometric functions like sin(), cos(), and tan().
  - Logarithmic and exponential functions like log(), exp().
- Logical Expressions:
  - Logical operators like and, or, and not.
  - Comparison operators like ==, !=, >, <, >=, and <=.
- Variable Assignment and Usage:
  - You can define variables and use them in expressions.
- List and Array Operations:
  - Allows for operations on lists and arrays, useful for numerical computations.
- Function Definition and Calls:
  - Define your functions and call them within expressions.
- Conditional Expressions:
  - Basic if-else conditional expressions.
- Loops: While loops, etc
- Built-in Functions and Constants:
  - Supports common built-in functions and mathematical constants like pi.

Random Numbers and Dice Rolling:

Randomized Selections (with weights):
${PICK|<item1>|>weight1>|<item2>|<weight2>|...|<itemN>|<weightN>}$

Example:
${PICK|Sunday|4|Monday|1|Tuesday|2|Wednesday|4|Thursday|1|Friday|7|Saturday|10}$

- This will randomly pick a day of the week; with Monday and Thursday being the least likely & Saturday the most likely (do to the weights attached to each selection)

Dice Rolling:
${ROLL|2|6|2}$

- This will roll 2d6+2 and return the total of the roll.

${ROLL_EVEN|6}$

- Will roll 6d6 and count the dice that rolled an even number.

Convert a stored table into a json string

As_json

Use this command when you want to turn one of your stored tables (like an inventory or lookup list) into a JSON string — a format that can be sent to a web server or another system.

Usage: ${as_json|<type>|<table_name>$}

Where <type> is either inventory, table, or include (include just sends a raw txt file)
And the <table_name> is the actual name of the document

Example:
${as_json|inventory|TreasureTable}$

This will convert the "Treasure Table" into a json string and send it to your endpoint, for example your web server.

LLM Function

Use this command if you want the AI to generate a response based on a custom prompt you give it. For example, you could ask your avatar to rewrite text or summarize a conversation.

Usage:

${LLM|<your prompt>|timeout=<seconds>}$

timeout (this sets how long to wait for the avatar's reply). Note, for executing summaries, we recommend using timeout=60.

Examples:

${LLM|Can you give me a dramatic and hilarious retelling in movie script style of the following conversation? "{{MIL_CHAT_LOG}}"|timeout=60}$

${LLM|Please summarize the conversation with the user"{{MIL_CHAT_LOG}}"|timeout=60}$

Echo Back Text in Prompts

Use this command if you want to return a fixed string of text exactly as you typed it. This is helpful for testing, inserting placeholder text, or outputting static messages in dynamic prompts.

Usage:

${ECHO|<your text>}$

Examples:

${ECHO|I'm not sure if this command is really needed.}$

${ECHO|Welcome to your session! Let's get started.}$

Changing Languages

You can change the language your Avatar speaks and understands by setting the MIL_LANGUAGE variable. This is especially helpful for multilingual users or language tutor avatars.

To change languages directly in your prompt, use: ${SET|MIL_LANGUAGE|ES}$ (This example sets the language to Spanish)

You can also teach your Avatar to change languages on its own. The avatar will automatically detect if the user is speaking to them in a supported language and

"When a user wants to speak in another language, use [[SET|MIL_LANGUAGE|LANGUAGE_CODE]]to switch languages."

Supported Languages

Your Avatar supports many languages from around the world. Here are some of the supported languages with their codes:

- 🇺🇸 English (USA) - EN
- 🇬🇧 English (UK) - GB
- 🇦🇺 English (Australia) - AU
- 🇨🇦 English (Canada) - CA
- 🇯🇵 Japanese - JP
- 🇨🇳 Chinese - ZH
- 🇩🇪 German - DE
- 🇮🇳 Hindi - HI
- 🇫🇷 French (France) - FR
- 🇨🇦 French (Canada) - FR-CA
- 🇰🇷 Korean - KO

- 🇧🇷 Portuguese (Brazil) - PT-BR
- 🇵🇹 Portuguese (Portugal) - PT
- 🇮🇹 Italian - IT
- 🇪🇸 Spanish (Spain) - ES
- 🇲🇽 Spanish (Mexico) - ES-MX
- 🇮🇩 Indonesian - ID
- 🇳🇱 Dutch - NL
- 🇹🇷 Turkish - TR
- 🇵🇭 Filipino - TL
- 🇵🇱 Polish - PL
- 🇸🇪 Swedish - SV
- 🇧🇬 Bulgarian - BG
- 🇷🇴 Romanian - RO
- 🇸🇦 Arabic (Saudi Arabia) - AR-SA
- 🇦🇪 Arabic (UAE) - AR-AE
- 🇨🇿 Czech - CS
- 🇬🇷 Greek - EL
- 🇫🇮 Finnish - FI
- 🇭🇷 Croatian - HR
- 🇲🇾 Malay - MS
- 🇸🇰 Slovak - SK
- 🇩🇰 Danish - DA
- 🇮🇳 Tamil - TA
- 🇺🇦 Ukrainian - UK
- 🇷🇺 Russian - RU
- 🇭🇺 Hungarian - HU
- 🇳🇴 Norwegian - NO
- 🇻🇳 Vietnamese - VI

When your Avatar switches languages, it will not only respond in that language but will also better understand user messages in that language.

Session Hooks

You can make your Avatar respond to key moments in a conversation — like when someone starts talking to it for the first time or when the conversation ends. These key moments are called “events,” and you can use Session Hooks to tell the Avatar what to do when they happen.

Think of Session Hooks like little instructions you attach to key points in a conversation. When one of these events happens, your Avatar can either, say something automatically or perform an internal action without saying anything out loud.

Here’s a basic example:

```
<session_hooks>
  <hook trigger="first_session" inject>The user has just started the conversation.</hook>
</session_hooks>
```

Each session hook has two main commands

- trigger: this tells the system *when* to run your hook
- inject: this determines *what happens* when the hook runs

Trigger - available options:

- first_session – this runs the very first time a user speaks to an avatar that has memory turned on.
- session_start – this runs when the user starts a new session with the avatar.
- turn_start – this runs when the avatar starts speaking
- turn_end – this runs when the avatar finishes speaking.
- session_end – this runs at the end of a conversation.

Inject -

- If inject is included – The text inside the hook is inserted directly into the Avatar's prompt
- If inject is not included – The system processes the contents as actions (like logging data) but doesn't display them to the user.

You can include more than one hook — even for the same event. Here's an example:

```
<session_hooks>
  <hook trigger="first_session" inject>This session was started on ${DateTime}$</hook>
  <hook trigger="session_start">${web_post|url|session started on ${datetime}$}$</hook>
  <hook trigger="turn_start" inject>Turn {{MIL_CURRENT_TURN}} has started</hook>
  <hook trigger="turn_end">${web_post|url|{{MIL_CURRENT_TURN}}}$</hook>
  <hook trigger="session_end">${web_post|url|session data goes here}$</hook>
</session_hooks>
```

Note: Any session hook you include will run when that trigger event happens — you can add as many as you want for each event.

When the Avatar runs a hook with injected text, that content is inserted into the system prompt just before the user's message is processed. This makes it a seamless part of the conversation flow, even if it's invisible to the user.

Client Events System

Website-to-Avatar Communication

The Client Events System allows your website and Avatar to talk to each other. Think of it as passing notes back and forth. Your website can send special messages to the Avatar, and the Avatar can send messages back to your website. This system enables powerful interactions like having your Avatar navigate users to different pages on your website or respond to page changes.

***Note: In order for client events to work, you will need to set up a Single Page Application (SPA) website.

When your website needs to tell the Avatar to do something, it can send any of these messages to do the following:

Control UI elements (ShowHUD, ShowChatLog, ShowSettings)

ShowHUD: Shows (true) or hides (false) the bottom bar in the app
[$RECEIVE_CLIENT_EVENT|ShowHUD|true]
[$RECEIVE_CLIENT_EVENT|ShowHUD|false]

ShowChatLog: Opens (true) or closes (false) the chat history window
[$RECEIVE_CLIENT_EVENT|ShowChatLog|true]
[$RECEIVE_CLIENT_EVENT|ShowChatLog|false]

ShowSettings: Opens (true) or closes (false) the settings window
[$RECEIVE_CLIENT_EVENT|ShowSettings|true]
[$RECEIVE_CLIENT_EVENT|ShowSettings|false]

Simulate user actions (TalkPressed, SendChat, QueueChat)

TalkPressed: Acts as if someone pressed (true) or released (false) the talk button
[$RECEIVE_CLIENT_EVENT|TalkPressed|true]
[$RECEIVE_CLIENT_EVENT|TalkPressed|false]

SendChat: Sends text as if a user typed it
[$RECEIVE_CLIENT_EVENT|SendChat|message text]
Example: Send "Hello Avatar" with [$RECEIVE_CLIENT_EVENT|SendChat|Hello Avatar]

QueueChat: Adds text that will be included with the user's next message
[$RECEIVE_CLIENT_EVENT|QueueChat|message text]
Example: Add "continuing from before" with
[$RECEIVE_CLIENT_EVENT|QueueChat|continuing from before]

Manage the conversation (EndCall)

EndCall: Ends the current conversation
[$RECEIVE_CLIENT_EVENT|EndCall]
Example: Simply use [$RECEIVE_CLIENT_EVENT|EndCall] without any extra information

Configure display options (AllowShowImages, AllowShowURLs)

AllowShowImages: Enables (true) or disables (false) showing images
[$RECEIVE_CLIENT_EVENT|AllowShowImages|true] or
[$RECEIVE_CLIENT_EVENT|AllowShowImages|false]

AllowShowURLs: Enables (true) or disables (false) showing clickable links
[$RECEIVE_CLIENT_EVENT|AllowShowURLs|true] or
[${RECEIVE_CLIENT_EVENT|AllowShowURLs|false]

Switch communication modes (SetModeVideo, SetModeAudio, SetModeText)

SetModeVideo: Switches to video chat mode
[$RECEIVE_CLIENT_EVENT|SetModeVideo]

SetModeAudio: Switches to audio chat mode
[$RECEIVE_CLIENT_EVENT|SetModeAudio]

SetModeText: Switches to text chat mode
[$RECEIVE_CLIENT_EVENT|SetModeText]

Avatar-to-Website Communication

Your Avatar can also send messages to your website. These messages can tell your website what's happening or ask it to do something:

Conversation status updates (ChatMessageStart,ChatMessageStream, ChatMessageEnd)

ChatMessageStart: Tells the website when the Avatar starts talking
[$SEND_CLIENT_EVENT|ChatMessageStart|{"speaker":"avatar","turn":1,"text":"Hello there"}]
The website gets details about who's speaking and what they're saying in JSON format

ChatMessageStream: Updates the website as the Avatar speaks each line
[$SEND_CLIENT_EVENT|ChatMessageStream|{"speaker":"avatar","turn":1,"line":2,"text":"How can I help?"}]

Helps your website show text as it appears rather than all at once

ChatMessageEnd: Tells the website when the Avatar is done responding
[$SEND_CLIENT_EVENT|ChatMessageEnd|{"speaker":"avatar","turn":1}]
Useful for knowing when to enable user input again

Session management (ChatSessionBegin, ChatSessionEnd)

ChatSessionBegin: Signals that the conversation is ready to start
[$SEND_CLIENT_EVENT|ChatSessionBegin]
Good time to show welcome information or instructions

ChatSessionEnd: Notifies that the conversation has ended
[$SEND_CLIENT_EVENT|ChatSessionEnd|{"reason":"user_left"}]
Includes a reason (user left, system timeout, etc.)

UI control requests (ImageShow, URLShow)

ImageShow: Asks the website to display an image
[$SEND_CLIENT_EVENT|ImageShow|{"url":"https://example.com/image.jpg"}]
Includes the image URL to show

URLShow: Asks the website to display a clickable link
[$SEND_CLIENT_EVENT|URLShow|{"url":"https://example.com","display_text":"Visit Our Website"}]
Includes both the link address and the text to display

System status information (MicAvailable, LaunchFailed, TalkEnabled)
MicAvailable: Reports whether the microphone is working
[$SEND_CLIENT_EVENT|MicAvailable|{"mic_name":"Default Microphone","available":true}]
Helps troubleshoot audio issues

LaunchFailed: Indicates the chat couldn't start
[$SEND_CLIENT_EVENT|LaunchFailed|{"reason":"microphone_not_available"}]
Includes the reason why it failed

TalkEnabled: Tells the website whether the talk button should be active
[$SEND_CLIENT_EVENT|TalkEnabled|true}$ or ${SEND_CLIENT_EVENT|TalkEnabled|false]
The website needs to respond to this message to keep things in sync

These messages are sent as structured data (JSON format) that your website can easily understand and process. Using this system, your Avatar and website can work together seamlessly to create interactive experiences. You can use custom events to build powerful navigation flows, allowing your Avatar to guide users through your website based on their conversation.

Custom Events

Your Avatar can send custom events that you create. This is especially helpful if you want your avatar to follow a user across webpages.

For example:

- Format: [$CLIENT_EVENT:EventName|Optional Extra Information]
- Example: [$CLIENT_EVENT:ShowConfetti|Birthday] triggers a confetti animation

Example: [$CLIENT_EVENT:NavigateTo|/products] changes to your products page.