



ENGLISH DOCUMENTATION

Model Vista MCP Module

Connect a local AI assistant to Model Vista

30 October 2026

Contents

01 MCP Module

02 Connect your AI client

03 Load models and choose States

04 Work with views and XR

05 Application tool reference

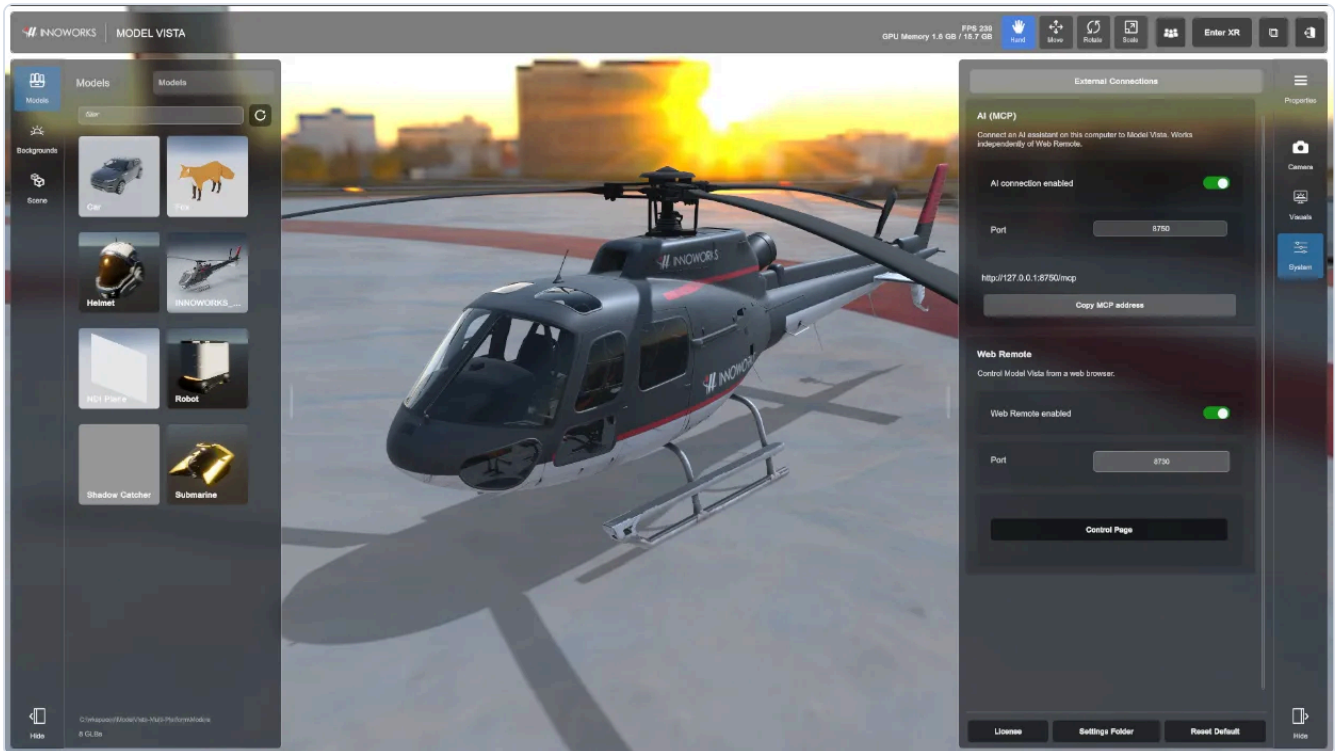
06 Discover additional runtime controls

07 Troubleshoot the MCP connection

MCP Module

The Model Vista MCP Module is a separately licensed Module that connects an AI assistant to a running Model Vista application through the Model Context Protocol (MCP). Ask the assistant to inspect models, select Model Vista States, adjust the scene, or capture the current view.

The application labels the connection **AI (MCP)** under **System > External Connections**. It works independently of **Web Remote**.



Requirements

Compatible Model Vista Versions

Application	Compatible versions
Model Vista for Windows, HDRP desktop and XR	2.0.6 with the MCP Module

Additional Requirements

- An MCP Module entitlement enabled under **System > License**, where it is listed as **AI (MCP)**.
- An AI client with Streamable HTTP MCP support, running on the same computer as Model Vista.
- A running Model Vista session and access to the models and backgrounds you want to review.

The MCP Module does not grant licenses for other Modules. Their controls remain subject to the corresponding entitlement and runtime availability.

Quick Start

1. [Enable the connection and configure your AI client.](#)
2. Ask the assistant to list loaded models without changing the scene.
3. [Load a model and choose a State](#) using names returned by discovery.
4. [Work with views and XR](#) or capture the current application view.
5. Check the returned state after each requested change.

Use the [application tool reference](#) for common tasks, the [Variable Exposer reference](#) for additional runtime controls, and [troubleshooting](#) for connection or operation failures.

Connect your AI client

Enable AI (MCP)

1. Start Model Vista.
2. Open **System > License** and confirm the **AI (MCP)** entitlement is available and enabled.
3. Open **System > External Connections**.
4. In **AI (MCP)**, turn on **AI connection enabled**.
5. Wait for the address to appear, then select **Copy MCP address**.

The default address is:

```
http://127.0.0.1:8750/mcp
```

The connection is disabled by default. Its enable setting and port are saved independently of Web Remote. A missing entitlement hides the AI (MCP) panel; enabling the connection cannot bypass the license requirement.

Add the server to your client

1. Open your AI client's MCP server settings.
2. Add a server named **Model Vista** using the **Streamable HTTP** transport.
3. Paste the copied address as the server URL.
4. Connect and enable the Model Vista tools for the conversation.
5. Ask: "Using Model Vista MCP, list the loaded models. Do not change anything."

A connected client should discover tools such as `list_loaded_models` and `variable_exposer_list_fields`. An empty loaded-model list means the scene has no loaded models; it does not mean the connection failed.

The listener accepts connections on the local computer. Run the client or its local connection component on that computer. The MCP address is separate from the Web Remote **Control Page** address.

A cloud-backed AI client may send tool results and screenshots to its model service. Choose a client configuration appropriate for your model content.

Change the port or disconnect

Enter an unused **Port** from `1` to `65535` . Changing the port restarts the listener; copy the new address and reconnect the client. Give simultaneous application instances different ports.

Turn off **AI connection enabled** to stop accepting MCP calls. Disabling AI (MCP) in **License** also stops the listener. Stopping the connection does not cancel an application operation that has already started.

Launch with a temporary setting

Launch arguments override the saved connection setting for that launch:

Terminal window

```
& '.\Model Vista.exe' --modelvista-mcp --modelvista-mcp-port=8750
```

Use `--modelvista-no-mcp` to start with MCP disabled. These arguments do not change saved preferences and still require the MCP Module entitlement.

Load models and choose States

Find and load a model

1. Ask the assistant to call `list_library_models` and show the available model names.
2. Choose one result and ask it to load that model with `load_model`.
3. Wait for loading to finish. Do not submit the same load again while it is pending.
4. Ask it to call `list_loaded_models` and identify the new scene instance.

Library asset IDs identify files that can be loaded. Loaded-model IDs identify instances already in the scene. Use the appropriate returned ID for each operation. If multiple models have the same name, use their IDs.

`list_libraries` and `select_library` change the library being browsed. Selecting a library does not load or remove scene models.

Inspect and apply a State

1. Use `list_loaded_models` to choose the model.
2. Call `list_model_states` for that loaded-model ID.
3. Choose a returned group and option, then ask the assistant to apply them with `set_model_state`.
4. Check `applied` and the selected-option readback, then inspect the visible result.

For example, ask: "List this model's State groups and options. Show me the choices before changing anything." After choosing an option, ask: "Apply that option and report the selected State afterward."

Arguments to `set_model_state` use strings:

```
{
  "model": "<loaded-model ID from list_loaded_models>",
  "group": "<group ID or exact name from list_model_states>",
  "option": "<option ID or exact name from list_model_states>"
}
```

State groups are authored model options, such as visibility, materials, animation, or named views. A standard GLB may have no Model Vista States. Discover the available options instead of assuming every model has doors, rotors, or camera views.

Adjust the scene

Read `get_model` before changing visibility or transforms. `set_model_transform` uses world-space metres for position, world-space degrees for rotation, and local scale. Relative scale multiplies each axis. Transform locks remain active unless you explicitly request `unlock: "true"`.

Use `list_backgrounds` before `set_background`. Read `get_environment` or `get_graphics_options` before changing their settings, and check the returned availability and applied state.

`delete_model` removes a scene instance and keeps its source library file. Name the intended instance explicitly before asking for removal, then refresh the loaded-model list.

Work with views and XR

Frame a desktop view

1. Ask the assistant to read `get_xr_mode`.
2. In desktop mode, choose a model from `list_loaded_models`.
3. Request `set_camera_view` with `front`, `back`, `left`, `right`, `top`, or `bottom`.
4. Read `get_camera` and inspect the result.

These views follow the model's local axes: +Z is front, +X is right, and +Y is top. They do not infer which part of a model is its semantic front.

Use `frame_model` to fit the model while preserving the current viewing direction. It starts an animated camera movement; an accepted request does not mean the camera has reached its final pose.

`move_camera` uses metre offsets relative to the current camera direction. `rotate_camera` changes the camera orientation or orbits its current focus. `set_camera_fov` accepts a vertical field of view from 15 to 90 degrees.

Use MCP during XR review

1. Read `get_xr_mode` and confirm that XR is available.
2. If you want to enter XR, request `set_xr_mode` with `enabled: "true"`.
3. Wait until readback confirms the requested mode and no transition is pending.
4. Use model, State, environment, and runtime UI tools as available during the review.

Desktop camera positioning tools are unavailable in XR. They do not reposition the headset user. Ask for a model State when you want an authored named view, and specify when the assistant should keep XR mode and model transforms unchanged.

To leave XR, request `set_xr_mode` with `enabled: "false"` and check the mode again before using desktop camera controls.

Show controls and capture the view

Use `list_windows` to discover available runtime panels. `show_window` opens a panel; for example, `window: "system"` with `section: "external-connections"` opens the connection settings. Use `list_ui_scrolls` after opening a panel, then `scroll_ui` with a returned area ID.

Ask the assistant to call `capture_screenshot` when you want it to inspect the visible result. The capture includes the application UI; in XR it uses the desktop mirror.

- `save: "true"` is the default. It saves a full-resolution PNG under `{PersistentDataPath}/Screenshots` and returns an image preview with the saved path.
- `save: "false"` returns the preview without saving the full-resolution file.

The preview is compressed and can be smaller than the saved image. For animation, compare views over time; `get_model_animation` reads playback clocks but does not prove that motion is visibly correct.

Application tool reference

Discover the connected server's tools before use. Tool availability and successful operation depend on the running application, loaded content, active mode, and licensed features.

Application-tool arguments are strings, including numbers and booleans such as `"true"`, `"10"`, and `"0.5"`. Follow each tool's discovered schema for required and optional arguments. The [Variable Exposer tools](#) use their own schemas.

Libraries, models, and States

Tool	Use
<code>list_libraries</code> / <code>select_library</code>	Discover and choose a model or background library; selection changes browsing.
<code>list_library_models</code> / <code>load_model</code>	Discover source assets and load one by library asset ID or exact unique name.
<code>list_loaded_models</code> / <code>select_model</code>	Discover scene instances and select one for runtime controls.
<code>get_model</code>	Read visibility, transforms, selection, and locks.
<code>set_model_visibility</code>	Show or hide a loaded model, subject to normal visibility rules.
<code>set_model_transform</code>	Change <code>position</code> , <code>rotation</code> , or <code>scale</code> ; use <code>absolute</code> or <code>relative</code> mode.
<code>delete_model</code>	Remove a scene instance; the source library file is retained.
<code>list_model_states</code> / <code>set_model_state</code>	Discover State groups/options and apply one; check selected-option readback.
<code>get_model_animation</code>	Sample playback clocks without changing playback.

Camera and XR

Tool	Use
<code>get_camera</code>	Read the desktop camera pose and lens.
<code>set_camera_view</code>	Frame a model from one of six model-axis views.
<code>frame_model</code>	Begin fitting a model along the current viewing direction.
<code>move_camera</code> / <code>rotate_camera</code>	Move, look around, or orbit the desktop camera.
<code>set_camera_fov</code>	Set the vertical perspective field of view.
<code>get_xr_mode</code> / <code>set_xr_mode</code>	Read, enter, or exit XR and inspect transition state.

The camera controls require desktop mode. See [views](#) and [XR](#) for positioning and capture behavior.

Runtime UI and screenshots

Tool	Use
<code>list_windows</code> / <code>show_window</code>	Discover runtime panels and open one, optionally selecting a settings section.
<code>hide_window</code>	Hide a panel or <code>all</code> ; behavior follows the active interface layout.
<code>show_library</code>	Show <code>models</code> or <code>backgrounds</code> thumbnails.
<code>list_ui_scrolls</code> / <code>scroll_ui</code>	Discover visible scroll areas and scroll a current area ID.
<code>capture_screenshot</code>	Return an application image preview and optionally save a PNG.

Backgrounds, graphics, and Turntable

Tool	Use
<code>list_backgrounds</code> / <code>set_background</code>	Discover and load an environment by returned ID or exact unique name.
<code>get_environment</code> / <code>set_environment</code>	Read supported settings and adjust one within its returned range.
<code>get_graphics_options</code> / <code>set_graphics_option</code>	Discover supported graphics toggles and change one with actual readback.
<code>get_turntable</code>	Read assignment, playback, angle, speed, and availability.
<code>assign_model_to_turntable</code>	Assign a loaded model; inspect <code>accepted</code> and <code>confirmed</code> .
<code>release_turntable_model</code>	Remove the assignment while keeping the model loaded.
<code>set_turntable</code>	Set <code>play</code> , <code>reverse</code> , <code>pause</code> , <code>angle</code> , or <code>speed</code> . MCP speed values use degrees per second, from <code>0</code> to <code>30</code> .

Networked assignment and release can remain pending after acceptance. Read `get_turntable` until the actual assignment matches the request. Pause before setting an angle when you need stable readback.

Discover additional runtime controls

Use the Variable Exposer tools when a task needs a runtime control beyond the [application tools](#). They discover registered fields, functions, and instance bindings in the running session.

Stable discovery tools

Tool	Arguments and purpose
<code>variable_exposer_list_fields</code>	Optional <code>path</code> filter; list fields and their metadata.
<code>variable_exposer_get_field</code>	<code>identifier</code> , with <code>targetId</code> for an instance; read the value and available options.
<code>variable_exposer_set_field</code>	<code>identifier</code> , <code>value</code> , and an instance <code>targetId</code> when needed; write a mutable field.
<code>variable_exposer_list_functions</code>	List callable functions and their descriptors.
<code>variable_exposer_get_function</code>	<code>identifier</code> and an instance <code>targetId</code> when needed; inspect function metadata.
<code>variable_exposer_list_instances</code>	<code>identifier</code> ; discover target IDs and labels for that path.
<code>variable_exposer_invoke_function</code>	<code>identifier</code> , optional instance <code>targetId</code> , and <code>arguments</code> matching the discovered payload type.
<code>variable_exposer_get_job</code>	<code>jobId</code> ; read a background invocation's status and result.

Discover before changing a value

1. List fields and functions. Related controls can appear in different categories.
2. Inspect the field options or function descriptor for the required identifier and payload.
3. For an instance-bound path, call `variable_exposer_list_instances` and use its returned `instanceId` as `targetId`.
4. Read the current value, then request the intended change.
5. Read the resulting value or application state to confirm the outcome.

Do not guess identifiers or instance IDs. Repeat discovery after loading or deleting a model because its controls and State instances can change. A writable field is not necessarily represented by a separate named function.

Unlike application-tool string arguments, `value` and function `arguments` can be JSON scalars, arrays, or objects. Match the discovered schema; include `targetId` when the same path has multiple instances.

Follow a long-running function

Set `background: true` on `variable_exposer_invoke_function` to receive a `jobId` immediately. Keep that ID and poll `variable_exposer_get_job` until the status is `succeeded` or `failed`. Do not invoke the function again while its job is running.

Inspect both execution status and the returned result. A completed function can return `false`, and an accepted request can still have application work pending. Confirm the requested change through readback rather than treating transport success as proof.

Job IDs expire when the listener restarts, and completed jobs can be evicted from its bounded cache. Stopping MCP does not undo or cancel work already started. If an ID expires, inspect the actual application state before deciding whether to retry.

Troubleshoot the MCP connection

Symptom	Check
AI (MCP) panel is missing	Confirm that the running application includes the Module and that AI (MCP) is licensed and enabled under System > License .
"Unable to start MCP"	The port may already be in use. Choose an unused port, copy the new address, and reconnect.
Client cannot connect	Keep Model Vista running, enable AI connection enabled , and use the displayed <code>/mcp</code> URL with Streamable HTTP on the same computer.
Web Remote works but MCP does not	Check the independent AI (MCP) toggle and port. The Control Page URL is not the MCP URL.
No loaded models are returned	Use <code>list_library_models</code> to discover assets, then load a chosen result.
Model or State name is ambiguous	Discover again and use the returned ID instead of a shared name.
A tool or field is missing	Reconnect to refresh tools, then inspect application tools, fields, functions, and instances. Some controls require a loaded model or another entitlement.
Desktop camera command fails in XR	Keep XR and use available model/State controls, or intentionally exit XR and wait for readback before using camera tools.
Change is rejected or <code>applied</code> is false	Check busy state, transform locks, visibility rules, graphics availability, and license status. Read the actual state before retrying.
Turntable assignment is accepted but not confirmed	Read <code>get_turntable</code> ; network acceptance can precede the actual assignment.
State is selected but no animation is visible	Inspect <code>get_model_animation</code> and the rendered view. Selected-option readback and advancing clocks do not prove visible motion.
Screenshot times out	Make the application or Game view visible and allow a rendered frame, then retry.
Job ID is no longer found	The listener may have restarted or the completed job may have expired. Inspect application state before repeating the operation.

After a restart or license change

Reconnect your AI client after changing the port, restarting Model Vista, or restoring the MCP entitlement. Refresh discovery before using previously returned instance IDs.

If a long operation times out in the client, check whether it is still running or has completed in Model Vista. A client timeout is not evidence that the operation was cancelled.

Report an issue

Include the Model Vista version, AI client and version, selected MCP port, desktop/XR mode, exact tool name and arguments, returned error or status, and expected result. Describe whether the issue followed a model load, listener restart, license change, or XR transition. Remove credentials and private model information from shared logs and screenshots.