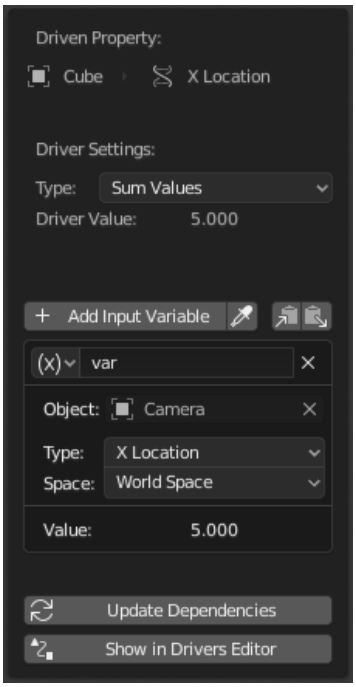
Skip to content

Drivers Panel



Edit Driver popover.

Reference

Editor:
Graph editor
Mode:
Drivers
Panel:
Sidebar region ▸ Drivers
Shortcut:
`N`

Reference

Menu:
Context menu ▸ Edit Driver
Shortcut:
`Ctrl - D`

This panel is visible in Sidebar of the Drivers Editor or as a popover when adding a driver to a property.

It shows the property that is being driven, followed by a series of settings that determine how the driver works.

Driver Settings

Type

There are two categories of drivers:

- **Built-in functions** (*Average*, *Sum*, *Min* and *Max*)

  The driven property will have the value of the average, sum, lowest or highest (respectively) of the values of the referenced *Driver Variables*. If there only one driver variable, these functions will yield the same result.

- **Custom** (*Scripted Expression*).

Custom (*Scripted Expression*):

An arbitrary Python expression that can refer to the *Driver Variables* by name. See Expressions.

Driver Value

The current result of the driver setup. Useful for debug purposes.

Variables

See Driver Variables.

Update Dependencies

Forces an update for the Driver Value dependencies.

Show in Drivers Editor

Opens the fully featured Drivers Editor. This button only appears in the popover version of the Drivers panel.

Driver Variables

Variables are references to properties, transformation channels, or the result of a comparison between transformations of two objects.

Drivers should access object data via *Driver Variables*, rather than direct references in the Python expression, in order for dependencies to be correctly tracked.

Add, Copy, Paste buttons.

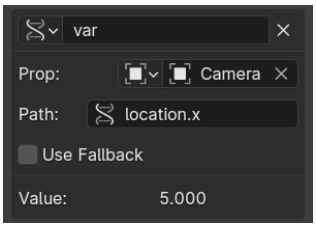
Add Input Variable
: Adds a new Driver Variable.

Copy/Paste Variables
: Copies the current variable list so it can be pasted into another driver's variable list.

Name
: Name for use in scripted expressions. The name must start with a letter, and only contain letters, digits, or underscores.

Variable Type
: The type of variable to use.

Single Property



Retrieves the value of an RNA property, specified by a data-block reference and a path string.

In case of transform properties, this will return the exact value of the UI property, while Transform Channel will take parenting and/or constraints into account as needed.

See also Custom Properties.

ID Type
: The ID-block type. For example: Key, Image, Object, Material.

ID
: The ID of the ID-block type. For example: "Material.001".

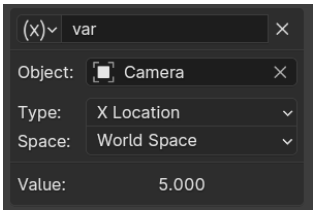
RNA Path
: The RNA name of the property, based on a subset of Python attribute access syntax. For example: `location.x` or `location[0]` for the X location animation channel value (before parenting or constraints), or `["prop_name"]` for a custo property.

Fallback
: If enabled, allows specifying a fallback value to use as the variable value if the RNA Path cannot be resolved, instead of causing a driver evaluation failure. For more info see Context Property below.

Tip

The easiest way to create a variable of this type is to use the Copy As New Driver context menu option of the input property, and paste the result into the driver via Paste Driver Variables.

Transform Channel



Retrieves the value of a Transform channel from an object or bone.

ID

ID of the object. For example: Cube, Armature, Camera.

Bone

For armatures, the name of the Armature bone. For example: "Bone", "Bone.002", "Arm.r".

Type

For example, X Location, X Rotation, X Scale.

The *Average Scale* option retrieves the combined scale value, computed as the cubic root of the total change in volume. Unlike *X/Y/Z Scale*, this value can be negative if the object is flipped by negative scaling.

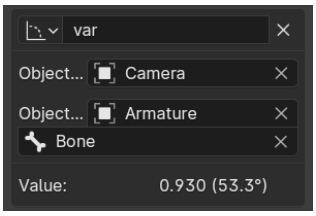
Mode (Rotation)

For rotation channels, specifies the type of rotation data to use, including different explicit Euler orders. Defaults to using the Euler order of the target. See Rotation Channel Modes.

Space

World Space, Transform Space, Local Space.

Rotational Difference

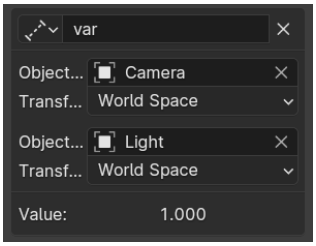


Provides the value of the rotational difference between two objects or bones, in radians.

Bone

For armatures, the name of the Armature bone. For example: "Bone", "Bone.002", "Arm.r".

Distance



Provides the value of the distance between two objects or bones.

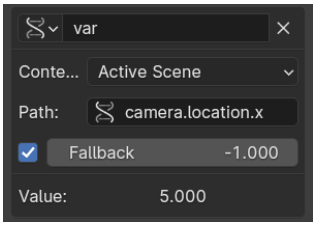
Bone

For armatures, the name of the Armature bone. For example: "Bone", "Bone.002", "Arm.r".

Space

World Space, Transform Space, Local Space.

Context Property



Provides the value of a property that is implicitly referring to either a scene or a view layer of the currently evaluating animation system. This is a weak reference which does not lead to the scene or view layer referenced from the driver to be linked when linking animation data.

An example when such properties comes in play is referring to a transformation of the active camera. It is possible to set up a driver in a character file, and make the driver use the set camera when the character is linked into a set.

Context

Active Scene, Active View Layer.

RNA Path

The RNA name of the property, based on a subset of Python attribute access syntax. For example: `camera.location.x` or `camera.location[0]` for the camera X location animation channel value (before parenting or constraints), or `["prop_name"]` for a custom property.

Fallback

If enabled, allows specifying a fallback value to use as the variable value if the RNA Path cannot be resolved, instead of causing a driver evaluation failure.

This feature can be very useful for making drivers more robust when implementing scene-global options using custom properties. Wh the object is linked into a different scene, these custom properties may not exist there, and the fallbacks can be used to provide sensible default values.

Fallbacks can also be used to emulate the lookup behavior of the View Layer mode of the material Attribute Node.

Tip

Although the values of the x/y/z animation channels for the camera location can be accessed via `camera.location[0/1/2]`, retrieving its world space location and orientation after parenting and constraints currently requires using `camera.matrix_world`. This property can be understood easily by viewing the matrix as an array of four vectors in *World* space:

- `matrix_world[0][0/1/2]` is the *Screen Right* direction vector (camera local X).
- `matrix_world[1][0/1/2]` is the *Screen Up* direction vector (camera local Y).
- `matrix_world[2][0/1/2]` is the **opposite** of the direction the camera is pointing.
- `matrix_world[3][0/1/2]` is the *location* of the camera.

Value

Shows the value of the variable.

Rotation Channel Modes

Rotation Transform Channels support a number of operation modes, including:

Auto Euler

Uses the Euler order of the target to decompose rotation into channels.

XYZ Euler, …

Explicitly specifies the Euler rotation order to use.

Quaternion

Provides the Quaternion representation of the rotation.

Swing and X/Y/Z Twist

Decomposes the rotation into two parts: a Swing rotation that aims the specified axis in its final direction, followed by a Twist rotation around that axis. This is often necessary for driving corrective Shape Keys and bones for organic joint rotation.

This decomposition is often produced in rigs by using a helper bone with a Damped Track Constraint to extract the swing part, and its child with Copy Transforms to extract the twist component.

The channels values for *Swing and Y Twist* are:

Y Rotation

True angle of the twist rotation.

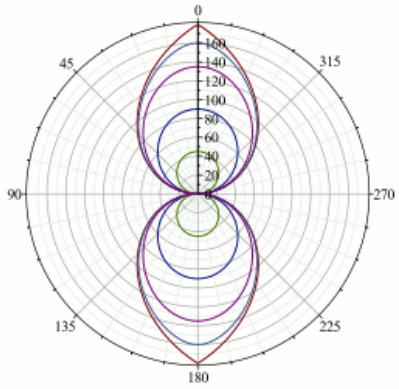
W Rotation

True angle of the swing rotation, independent of its direction.

X Rotation, Z Rotation

Weighted angles that represent the amount of swing around the X/Z axis.

The magnitude of the angle equals *W Rotation* when the rotation is purely around that axis, and fades out to zero as the direction changes toward the other axis, following the falloff curves from the graph on the right.



Falloff curves for weighted angles.

Mathematically, the swing angles are computed from quaternion components, using $2 \arccos(w)$ for *W* and $2 \arcsin(x)$ etc. for the others. The component of the swing rotation that corresponds to the twist axis is always 0, and is replaced by the twist angle.

Expressions

Expression

Expression

A text field where you can enter an arbitrary Python expression that refers to *Driver Variables* by their names.

The expression has access to a set of standard constants and math functions from `math`, `bl_math` and other modules, provided in the *Drive Namespace*. For an example of adding a custom function to the namespace, see the driver namespace example.

For performance reasons it is best to use the Simple Expressions subset as much as possible.

Use Self

If this option is enabled, the variable `self` can be used for drivers to reference their own data. Useful for objects and bones to avoid having creating a *Driver Variable* pointing to itself.

Example: `self.location.x` applied to the Y rotation property of the same object will make the object tumble when moving.

Note that dependencies for properties accessed via `self` may not be fully tracked.

Simple Expressions

Blender can evaluate a useful subset of Python driver expressions directly, which significantly improves performance, especially on multi-core systems. T take advantage of this, the driver expression must only use the following features:

Variable Names

Use only ASCII characters.

Literals

Floating-point and decimal integer.

Globals

`frame`

Constants

`pi`, `True`, `False`

Operators

`+`, `-`, `*`, `/`, `==`, `!=`, `<`, `<=`, `>`, `>=`, `and`, `or`, `not`, conditional operator/ ternary if

Standard Functions

`min`, `max`, `radians`, `degrees`, `abs`, `fabs`, `floor`, `ceil`, `trunc`, `round`, `int`, `sin`, `cos`, `tan`, `asin`, `acos` `atan`, `atan2`, `exp`, `log`, `sqrt`, `pow`, `fmod`

Blender Provided Functions

`lerp`, `clamp`, `smoothstep`

Simple expressions are evaluated even when Python script execution is disabled.

When an expression outside of this subset is used, Blender displays a “Slow Python expression” warning. However, as long as the majority of drivers us simple expressions, using a complex expression in select few is OK.

See also

- Extending Blender with Python.
- Python and its documentation.
- functions.wolfram.com.

Made with Furo
Last updated on 2025-05-10