Skip to content

Data-Blocks

The base unit for any Blender project is the data-block. Examples of data-blocks include: meshes, objects, materials, textures, node trees, scenes, texts, brushes, and even Workspaces.

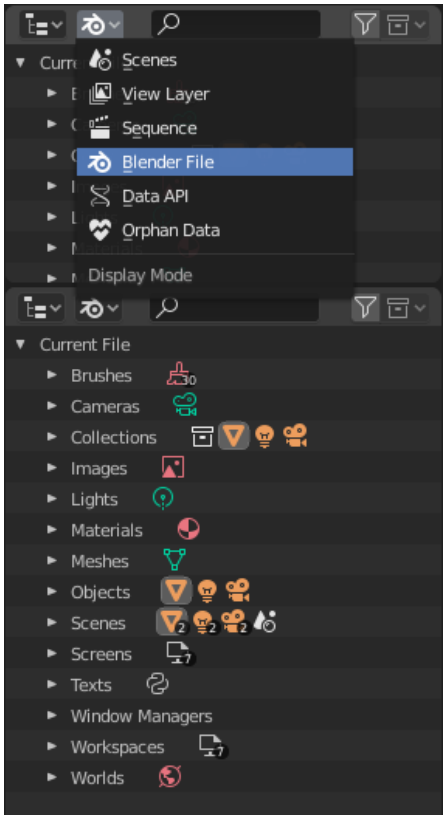
A data-block is a generic abstraction of very different kinds of data, which features a common set of basic features, properties and behaviors.

Some common characteristics:

- They are the primary contents of the blend-file.
- They can reference each other, for reuse and instancing. (Child/parent, object/object-data, materials/images, in modifiers or constraints too…)
- Their names are unique within a blend-file, for a given type.
- They can be added/removed/edited/duplicated.
- They can be linked between files (only enabled for a limited set of data-blocks).
- They can have their own animation data.
- They can have Custom Properties.

User will typically interact with the higher level data types (objects, meshes, etc.). When doing more complex projects, managing data-blocks becomes more important, especially when inter-linking blend-files. The main editor for that is the Outliner.

Not all data in Blender is a data-block, bones, sequence strips or vertex groups e.g. are not, they belong to armature, scene and mesh types respectively.



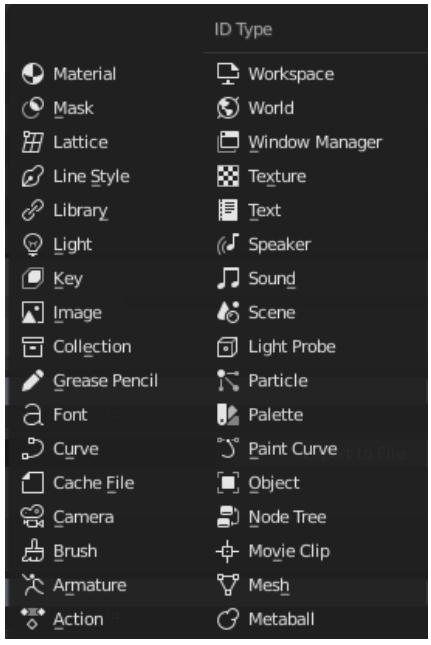
Blender File view of the Outliner.

Data-Block Types

For reference, here is a table of data-blocks types stored in blend-files.

Link
: Library Linking, supports being linked into other blend-files.

Pack
: File Packing, supports file contents being packed into the blend-file (**not** applicable for most data-blocks which have no file reference).



Data-blocks types with their icon.

Type	Link	Pack	Description
Action	✓	—	Stores animation F-Curves. Used as data-block animation data, and the Nonlinear Animation editor.
Armature	✓	—	Skeleton used to deform meshes. Used as data of armature objects, and by the Armature Modifier.
Brush	✓	—	Used as brush assets in sculpt and paint modes

Type	Link	Pack	Description
Camera	✓	—	Used as data by camera objects.
Cache File	✓	—	Used by Mesh Cache modifiers.
Curve	✓	—	Used as data by curve, font & surface objects.
Font	✓	✓	References font files. Used by curve object-data of text objects.
Grease Pencil	✓	—	2D/3D sketch data used by Grease Pencil objects. Used as overlay *helper* info, by the 3D Viewport, Image, Sequencer & Movie Clip editors.
Collection	✓	—	Group and organize objects in scenes. Used to instance objects, and in library linking.
Image	✓	✓	Image files. Used by shader nodes and textures.
Keys (Shape Keys)	✗	—	Geometry shape storage, which can be animated. Used by mesh, curve, and lattice objects.
Light	✓	—	Used as object data by light objects.
Library	✗	✓	References to an external blend-file. Access from the Outliner's *Blender File* view.
Line Style	✓	—	Used by the Freestyle renderer.
Lattice	✓	—	Grid based lattice deformation. Used as data of lattice objects, and by the Lattice Modifier.
Mask	✓	—	2D animated mask curves. Used by compositing nodes & sequencer strip.
Material	✓	—	Set shading and texturing render properties. Used by objects, meshes & curves.
Metaball	✓	—	An isosurface in 3D space. Used as data of metaball objects.
Mesh	✓	—	Geometry made of vertices/edges/faces. Used as data of mesh objects.
Movie Clip	✓	✗	Reference to an image sequence or video file. Used in the *Movie Clip* editor.
Node Tree	✓	—	Groups of re-usable nodes. Used in the node editors.
Object	✓	—	An entity in the scene with location, scale, rotation. Used by scenes & collections.
Paint Curve	✓	—	Stores a paint or sculpt stroke. Access from the paint tools.
Palette	✓	—	Store color presets. Access from the paint tools.
Particle	✓	—	Particle settings. Used by particle systems.
Light Probe	✓	—	Help achieve complex real-time lighting in EEVEE.
Scene	✓	—	Primary store of all data displayed and animated. Used as top-level storage for objects & animation.
Sounds	✓	✓	Reference to sound files. Used as data of speaker objects.
Speaker	✓	—	Sound sources for a 3D scene. Used as data of speaker object.
Text	✓	✗	Text data. Used by Python scripts and OSL shaders.
Texture	✓	—	2D/3D textures. Used by brushes and modifiers.
Window Manager	✗	—	The overarching manager for all of Blender's user interface. Includes Workspaces, notification system, operators, and keymaps.
World	✓	—	Define global render environment settings.
Workspace	✗	—	UI layout. Used by each window, which has its own workspace.

Life Time

Every data-block has its usage counted (reference count), when there is more than one, you can see the number of current users of a data-block to the right of its name in the interface. Blender follows the general rule that unused data is eventually removed.

Since it is common to add and remove a lot of data while working, this has the advantage of not having to manually manage every single data-block. This works by skipping zero user data-blocks when writing blend-files.

Protected

Since zero user data-blocks are not saved, there are times when you want to force the data to be kept irrespective of its users.

If you are building a blend-file to serve as a library of assets that you intend to link to and from other files, you will need to make sure that they do not accidentally get deleted from the library file.

To protect a data-block, use the button with the shield icon next to its name. The data-block will then never be silently deleted by Blender, but you can s manually remove it if needed.

> Note
>
> Linked data cannot be protected that way.

Name & Rename

Data-blocks names are unique within their namespace. A data-block namespace is defined by its type, and the blendfile it is stored in.

This means that there can be for example an Object and a Mesh named the same, but there cannot be two local objects named the same in a blendfile. However, it is possible to have one local and several linked Objects sharing the same name.

Data-block names have a fixed length of 63 bytes, i.e. 63 basic ASCII characters, or less when using diacritics or non-latin glyphs (the UTF8 encoding will then typically use more than a byte per character).

When Blender has to name a new data-block, or rename an existing one, it will check for name collisions. If a data-block with the same name already exists, the (re)named data-block will get a numeric extension added as a post-fix to its 'root name', like e.g. *.001*. The first available index is used (up to the *999* value, after that the postfix index values are simply incremented until no collision happen anymore).

In case adding the numeric suffix would make the data-block name too long, the root name part will be shortened as needed.

Blender will never rename another data-block when doing automatic naming. So e.g. when adding a new *Cube* object and there are already *Cube* and *Cube.001* local objects, the new one will be named *Cube.002*.

Local data-blocks can be renamed by the user in several places in the UI (like the ID selection widget, or the Outliner view). When renamed from the U the behavior in case of name collision is as follow:

- If the original root name is different than in the new requested name, the renamed data-block gets the first available numerical suffix.
  - E.g. assuming that there are three objects named *Sphere*, *Cube* and *Cube.001*, renaming *Sphere* to *Cube* will rename the data-block to *Cube.00*
- If the original root name is the same as in the new requested name, the renamed data-block gets the requested name, and the conflicting of data-block is renamed accordingly.
  - E.g. assuming that there are three objects named *Sphere*, *Cube* and *Cube.001*, renaming *Cube.001* to *Cube* will rename the data-block to *Cube* and the other data-block to *Cube.001*.

Sharing

Data-blocks can be shared among other data-blocks.

Examples where sharing data is common:

- Sharing textures among materials.
- Sharing meshes between objects (instances).
- Sharing animated actions between objects, for example to make all the lights dim together.

You can also share data-blocks between files, see linked libraries.

Making Single User

When a data-block is shared between several users, you can make a copy of it for a given user. To do so, click on the user count button to the right of it name. This will duplicate that data-block and assign the newly created copy to that usage only.

> Note
>
> Objects have a set of more advanced actions to become single-user, see their documentation.

Removing Data-Blocks

Removing Data-Blocks

As covered in Life Time, data-blocks are typically removed when they are no longer used. They can also be manually *unlinked* or *deleted*.

Unlinking a data-block means that its user won't use it anymore. This can be achieved by clicking on the "X" icon next to a data-block's name. If you unlink a data-block from all of its users, it will eventually be deleted by Blender as described above (unless it is a protected one).

Deleting a data-block directly erases it from the blend-file, automatically unlinking it from all of its users. This can be achieved by `Shift - LMB` on t "X" icon next to its name.

> Warning
>
> Deleting some data-blocks can lead to deletion of some of its users, which would become invalid without them. The main example is that object-data deletion (like mesh, curve, camera…) will also delete all objects using it.

Those two operations are also available in the context menu when `RMB` -clicking on a data-block in the *Outliner*.

Made with Furo
Last updated on 2025-05-10