Skip to content

BMesh Types (bmesh.types)

Base Mesh Type

class bmesh.types.**BMesh**

The BMesh data structure

calc_loop_triangles()

Calculate triangle tessellation from quads/ngons.

RETURNS:

The triangulated faces.

RETURN TYPE:

list[tuple[`BMLoop` , `BMLoop` , `BMLoop`]]

calc_volume(signed=False)

Calculate mesh volume based on face normals.

PARAMETERS:

signed (*bool*) – when signed is true, negative values may be returned.

RETURNS:

The volume of the mesh.

RETURN TYPE:

float

clear()

Clear all mesh data.

copy()

RETURNS:

A copy of this BMesh.

RETURN TYPE:

`BMesh`

free()

Explicitly free the BMesh data from memory, causing exceptions on further access.

Note

The BMesh is freed automatically, typically when the script finishes executing. However in some cases its hard to predict when this will be and its useful to explicitly free the data.

from_mesh(mesh, face_normals=True, vertex_normals=True, use_shape_key=False, shape_key_index=0)

Initialize this bmesh from existing mesh datablock.

PARAMETERS:

- **mesh** (`Mesh`) – The mesh data to load.
- **use_shape_key** (*bool*) – Use the locations from a shape key.
- **shape_key_index** (*int*) – The shape key index to use.

Note

Multiple calls can be used to join multiple meshes.

Custom-data layers are only copied from `mesh` on initialization. Further calls will copy custom-data to matching layers, layers missing on the target mesh won't be added.

from_object(object, depsgraph, cage=False, face_normals=True, vertex_normals=True)

Initialize this bmesh from existing object data-block (only meshes are currently supported).

PARAMETERS:

- **object** (`Object`) – The object data to load.
- **cage** (*bool*) – Get the mesh as a deformed cage.
- **face_normals** (*bool*) – Calculate face normals.
- **vertex_normals** (*bool*) – Calculate vertex normals.

normal_update()

Update normals of mesh faces and verts.

Note

The normal of any vertex where `is_wire` is True will be a zero vector.

select_flush(select)

Flush selection, independent of the current selection mode.

PARAMETERS:

select (*bool*) – flush selection or de-selected elements.

select_flush_mode()

flush selection based on the current mode current `BMesh.select_mode` .

to_mesh(mesh)

Writes this BMesh data into an existing Mesh datablock.

PARAMETERS:

mesh (`Mesh`) – The mesh data to write into.

transform(matrix, filter=None)

Transform the mesh (optionally filtering flagged data only).

PARAMETERS:

- **matrix** (`mathutils.Matrix`) – 4x4x transform matrix.
- **filter** (*set[str]*) – set of values in ('SELECT', 'HIDE', 'SEAM', 'SMOOTH', 'TAG').

edges

This meshes edge sequence (read-only).

TYPE:

`BMEdgeSeq`

faces

This meshes face sequence (read-only).

TYPE:

`BMFaceSeq`

is_valid

True when this element is valid (hasn't been removed).

TYPE:
bool

is_wrapped

True when this mesh is owned by blender (typically the editmode BMesh).

TYPE:
bool

loops

This meshes loops (read-only).

TYPE:
`BMLoopSeq`

Note

Loops must be accessed via faces, this is only exposed for layer access.

select_history

Sequence of selected items (the last is displayed as active).

TYPE:
`BMEditSelSeq`

select_mode

The selection mode, values can be {'VERT', 'EDGE', 'FACE'}, can't be assigned an empty set.

TYPE:
set

verts

This meshes vert sequence (read-only).

TYPE:
`BMVertSeq`

Mesh Elements

class bmesh.types.**BMVert**

The BMesh vertex type

calc_edge_angle(fallback=None)

Return the angle between this vert's two connected edges.

PARAMETERS:
fallback (*Any*) – return this when the vert doesn't have 2 edges (instead of raising a `ValueError`).

RETURNS:
Angle between edges in radians.

RETURN TYPE:
float

calc_shell_factor()

Return a multiplier calculated based on the sharpness of the vertex. Where a flat surface gives 1.0, and higher values sharper edges. This is use to maintain shell thickness when offsetting verts along their normals.

RETURNS:

offset multiplier

RETURN TYPE:

float

copy_from(other)

Copy values from another element of matching type.

copy_from_face_interp(face)

Interpolate the customdata from a face onto this loop (the loops vert should overlap the face).

PARAMETERS:

face (`BMFace`) – The face to interpolate data from.

copy_from_vert_interp(vert_pair, fac)

Interpolate the customdata from a vert between 2 other verts.

PARAMETERS:

vert_pair (Sequence[`BMVert`]) – The verts between which to interpolate data from.

hide_set(hide)

Set the hide state. This is different from the *hide* attribute because it updates the selection and hide state of associated geometry.

PARAMETERS:

hide (*bool*) – Hidden or visible.

normal_update()

Update vertex normal. This does not update the normals of adjoining faces.

Note

The vertex normal will be a zero vector if vertex `is_wire` is True.

select_set(select)

Set the selection. This is different from the *select* attribute because it updates the selection state of associated geometry.

PARAMETERS:

select (*bool*) – Select or de-select.

Note

Currently this only flushes down, so selecting a face will select all its vertices but de-selecting a vertex won't de-select all the faces that use it, before finishing with a mesh typically flushing is still needed.

co

The coordinates for this vertex as a 3D, wrapped vector.

TYPE:

`mathutils.Vector`

hide

Hidden state of this element.

TYPE:

bool

index

Index of this element.

TYPE:

int

Note

This value is not necessarily valid, while editing the mesh it can become *dirty*.

It's also possible to assign any number to this attribute for a scripts internal logic.

To ensure the value is up to date - see `BMElemSeq.index_update`.

is_boundary

True when this vertex is connected to boundary edges (read-only).

TYPE:

bool

is_manifold

True when this vertex is manifold (read-only).

TYPE:

bool

is_valid

True when this element is valid (hasn't been removed).

TYPE:

bool

is_wire

True when this vertex is not connected to any faces (read-only).

TYPE:

bool

link_edges

Edges connected to this vertex (read-only).

TYPE:

`BMElemSeq` of `BMEdge`

link_faces

Faces connected to this vertex (read-only).

TYPE:

`BMElemSeq` of `BMFace`

link_loops

Loops that use this vertex (read-only).

TYPE:

`BMElemSeq` of `BMLoop`

normal

The normal for this vertex as a 3D, wrapped vector.

TYPE:

mathutils.vector

select

Selected state of this element.

TYPE:

bool

tag

Generic attribute scripts can use for own logic

TYPE:

bool

class bmesh.types.**BMEdge**

The BMesh edge connecting 2 verts

calc_face_angle(fallback=None)

PARAMETERS:

fallback (*Any*) – return this when the edge doesn't have 2 faces (instead of raising a `ValueError`).

RETURNS:

The angle between 2 connected faces in radians.

RETURN TYPE:

float

calc_face_angle_signed(fallback=None)

PARAMETERS:

fallback (*Any*) – return this when the edge doesn't have 2 faces (instead of raising a `ValueError`).

RETURNS:

The angle between 2 connected faces in radians (negative for concave join).

RETURN TYPE:

float

calc_length()

RETURNS:

The length between both verts.

RETURN TYPE:

float

calc_tangent(loop)

Return the tangent at this edge relative to a face (pointing inward into the face). This uses the face normal for calculation.

PARAMETERS:

loop (`BMLoop`) – The loop used for tangent calculation.

RETURNS:

a normalized vector.

RETURN TYPE:

mathutils.Vector

copy_from(other)

Copy values from another element of matching type.

hide_set(hide)

hide_set(hide)

Set the hide state. This is different from the *hide* attribute because it updates the selection and hide state of associated geometry.

PARAMETERS:

hide (*bool*) – Hidden or visible.

normal_update()

Update normals of all connected faces and the edge verts.

Note

The normal of edge vertex will be a zero vector if vertex `is_wire` is True.

other_vert(vert)

Return the other vertex on this edge or None if the vertex is not used by this edge.

PARAMETERS:

vert (`BMVert`) – a vert in this edge.

RETURNS:

The edges other vert.

RETURN TYPE:

`BMVert` | None

select_set(select)

Set the selection. This is different from the *select* attribute because it updates the selection state of associated geometry.

PARAMETERS:

select (*bool*) – Select or de-select.

Note

Currently this only flushes down, so selecting a face will select all its vertices but de-selecting a vertex won't de-select all the faces that use it, before finishing with a mesh typically flushing is still needed.

hide

Hidden state of this element.

TYPE:

bool

index

Index of this element.

TYPE:

int

Note

This value is not necessarily valid, while editing the mesh it can become *dirty*.

It's also possible to assign any number to this attribute for a scripts internal logic.

To ensure the value is up to date - see `BMElemSeq.index_update` .

is_boundary

True when this edge is at the boundary of a face (read-only).

TYPE:

bool

bool

is_contiguous

True when this edge is manifold, between two faces with the same winding (read-only).

TYPE:

bool

is_convex

True when this edge joins two convex faces, depends on a valid face normal (read-only).

TYPE:

bool

is_manifold

True when this edge is manifold (read-only).

TYPE:

bool

is_valid

True when this element is valid (hasn't been removed).

TYPE:

bool

is_wire

True when this edge is not connected to any faces (read-only).

TYPE:

bool

link_faces

Faces connected to this edge, (read-only).

TYPE:

`BMElemSeq` of `BMFace`

link_loops

Loops connected to this edge, (read-only).

TYPE:

`BMElemSeq` of `BMLoop`

seam

Seam for UV unwrapping.

TYPE:

bool

select

Selected state of this element.

TYPE:

bool

smooth

Smooth state of this element.

TYPE:
bool

tag

Generic attribute scripts can use for own logic

TYPE:
bool

verts

Verts this edge uses (always 2), (read-only).

TYPE:
`BMElemSeq` of `BMVert`

class bmesh.types.**BMFace**

The BMesh face with 3 or more sides

calc_area()

Return the area of the face.

RETURNS:
Return the area of the face.

RETURN TYPE:
float

calc_center_bounds()

Return bounds center of the face.

RETURNS:
a 3D vector.

RETURN TYPE:
`mathutils.Vector`

calc_center_median()

Return median center of the face.

RETURNS:
a 3D vector.

RETURN TYPE:
`mathutils.Vector`

calc_center_median_weighted()

Return median center of the face weighted by edge lengths.

RETURNS:
a 3D vector.

RETURN TYPE:
`mathutils.Vector`

calc_perimeter()

Return the perimeter of the face.

RETURNS:
Return the perimeter of the face.

RETURN TYPE:

float

calc_tangent_edge()

Return face tangent based on longest edge.

RETURNS:

a normalized vector.

RETURN TYPE:

mathutils.Vector

calc_tangent_edge_diagonal()

Return face tangent based on the edge farthest from any vertex.

RETURNS:

a normalized vector.

RETURN TYPE:

mathutils.Vector

calc_tangent_edge_pair()

Return face tangent based on the two longest disconnected edges.

- Tris: Use the edge pair with the most similar lengths.
- Quads: Use the longest edge pair.
- NGons: Use the two longest disconnected edges.

RETURNS:

a normalized vector.

RETURN TYPE:

mathutils.Vector

calc_tangent_vert_diagonal()

Return face tangent based on the two most distant vertices.

RETURNS:

a normalized vector.

RETURN TYPE:

mathutils.Vector

copy(verts=True, edges=True)

Make a copy of this face.

PARAMETERS:

- **verts** (*bool*) – When set, the faces verts will be duplicated too.
- **edges** (*bool*) – When set, the faces edges will be duplicated too.

RETURNS:

The newly created face.

RETURN TYPE:

BMFace

copy_from(other)

Copy values from another element of matching type.

copy_from_face_interp(face, vert=True)

Interpolate the customdata from another face onto this one (faces should overlap).

PARAMETERS:

- **face** (`BMFace`) – The face to interpolate data from.
- **vert** (*bool*) – When True, also copy vertex data.

hide_set(hide)

Set the hide state. This is different from the *hide* attribute because it updates the selection and hide state of associated geometry.

PARAMETERS:

hide (*bool*) – Hidden or visible.

normal_flip()

Reverses winding of a face, which flips its normal.

normal_update()

Update face normal based on the positions of the face verts. This does not update the normals of face verts.

select_set(select)

Set the selection. This is different from the *select* attribute because it updates the selection state of associated geometry.

PARAMETERS:

select (*bool*) – Select or de-select.

Note

Currently this only flushes down, so selecting a face will select all its vertices but de-selecting a vertex won't de-select all the faces that use it, before finishing with a mesh typically flushing is still needed.

edges

Edges of this face, (read-only).

TYPE:

`BMElemSeq` of `BMEdge`

hide

Hidden state of this element.

TYPE:

bool

index

Index of this element.

TYPE:

int

Note

This value is not necessarily valid, while editing the mesh it can become *dirty*.

It's also possible to assign any number to this attribute for a scripts internal logic.

To ensure the value is up to date - see `BMElemSeq.index_update` .

is_valid

True when this element is valid (hasn't been removed).

TYPE:
bool

loops
Loops of this face, (read-only).

TYPE:
`BMElemSeq` of `BMLoop`

material_index
The face's material index.

TYPE:
int

normal
The normal for this face as a 3D, wrapped vector.

TYPE:
`mathutils.Vector`

select
Selected state of this element.

TYPE:
bool

smooth
Smooth state of this element.

TYPE:
bool

tag
Generic attribute scripts can use for own logic

TYPE:
bool

verts
Verts of this face, (read-only).

TYPE:
`BMElemSeq` of `BMVert`

class bmesh.types.**BMLoop**
This is normally accessed from `BMFace.loops` where each face loop represents a corner of the face.

calc_angle()
Return the angle at this loops corner of the face. This is calculated so sharper corners give lower angles.

RETURNS:
The angle in radians.

RETURN TYPE:
float

calc_normal()

Return normal at this loops corner of the face. Falls back to the face normal for straight lines.

RETURNS:

a normalized vector.

RETURN TYPE:

`mathutils.Vector`

calc_tangent()

Return the tangent at this loops corner of the face (pointing inward into the face). Falls back to the face normal for straight lines.

RETURNS:

a normalized vector.

RETURN TYPE:

`mathutils.Vector`

copy_from(other)

Copy values from another element of matching type.

copy_from_face_interp(face, vert=True, multires=True)

Interpolate the customdata from a face onto this loop (the loops vert should overlap the face).

PARAMETERS:

- **face** (`BMFace`) – The face to interpolate data from.
- **vert** (*bool*) – When enabled, interpolate the loops vertex data (optional).
- **multires** (*bool*) – When enabled, interpolate the loops multires data (optional).

edge

The loop's edge (between this loop and the next), (read-only).

TYPE:

`BMEdge`

face

The face this loop makes (read-only).

TYPE:

`BMFace`

index

Index of this element.

TYPE:

int

Note

This value is not necessarily valid, while editing the mesh it can become *dirty*.

It's also possible to assign any number to this attribute for a scripts internal logic.

To ensure the value is up to date - see `BMElemSeq.index_update`.

is_convex

True when this loop is at the convex corner of a face, depends on a valid face normal (read-only).

TYPE:

bool

is_valid

True when this element is valid (hasn't been removed).

TYPE:

bool

link_loop_next

The next face corner (read-only).

TYPE:

BMLoop

link_loop_prev

The previous face corner (read-only).

TYPE:

BMLoop

link_loop_radial_next

The next loop around the edge (read-only).

TYPE:

BMLoop

link_loop_radial_prev

The previous loop around the edge (read-only).

TYPE:

BMLoop

link_loops

Loops connected to this loop, (read-only).

TYPE:

BMElemSeq of BMLoop

tag

Generic attribute scripts can use for own logic

TYPE:

bool

vert

The loop's vertex (read-only).

TYPE:

BMVert

Sequence Accessors

class bmesh.types.**BMElemSeq**

General sequence type used for accessing any sequence of BMVert, BMEdge, BMFace, BMLoop.

When accessed via BMesh.verts, BMesh.edges, BMesh.faces there are also functions to create/remove items.

index_update()

Initialize the index values of this sequence.

This is the equivalent of looping over all elements and assigning the index values.

```
for index, ele in enumerate(sequence):
    ele.index = index
```

Note

Running this on sequences besides `BMesh.verts`, `BMesh.edges`, `BMesh.faces` works but won't result in each element having a valid index, instead its order in the sequence will be set.

class bmesh.types.**BMVertSeq**

ensure_lookup_table()

Ensure internal data needed for int subscription is initialized with verts/edges/faces, eg `bm.verts[index]`.

This needs to be called again after adding/removing data in this sequence.

index_update()

Initialize the index values of this sequence.

This is the equivalent of looping over all elements and assigning the index values.

```
for index, ele in enumerate(sequence):
    ele.index = index
```

Note

Running this on sequences besides `BMesh.verts`, `BMesh.edges`, `BMesh.faces` works but won't result in each element having a valid index, instead its order in the sequence will be set.

new(co=(0.0, 0.0, 0.0), example=None)

Create a new vertex.

PARAMETERS:

- **co** (*float triplet*) – The initial location of the vertex (optional argument).
- **example** (`BMVert`) – Existing vert to initialize settings.

RETURNS:

The newly created vertex.

RETURN TYPE:

`BMVert`

remove(vert)

Remove a vert.

sort(key=None, reverse=False)

Sort the elements of this sequence, using an optional custom sort key. Indices of elements are not changed, `BMElemSeq.index_update` can be used for that.

PARAMETERS:

- **key** (Callable[[`BMVert` | `BMEdge` | `BMFace`], int] | None) – The key that sets the ordering of the elements.
- **reverse** (*bool*) – Reverse the order of the elements

Note

When the 'key' argument is not provided, the elements are reordered following their current index value. In particular this can be used by setting indices manually before calling this method.

Warning

Existing references to the N'th element, will continue to point the data at that index.

layers

custom-data layers (read-only).

TYPE:

BMLayerAccessVert

class bmesh.types.**BMEdgeSeq**

ensure_lookup_table()

Ensure internal data needed for int subscription is initialized with verts/edges/faces, eg `bm.verts[index]`.

This needs to be called again after adding/removing data in this sequence.

get(verts, fallback=None)

Return an edge which uses the **verts** passed.

PARAMETERS:

- **verts** (Sequence[BMVert]) – Sequence of verts.
- **fallback** – Return this value if nothing is found.

RETURNS:

The edge found or None

RETURN TYPE:

BMEdge

index_update()

Initialize the index values of this sequence.

This is the equivalent of looping over all elements and assigning the index values.

```
for index, ele in enumerate(sequence):
    ele.index = index
```

Note

Running this on sequences besides `BMesh.verts`, `BMesh.edges`, `BMesh.faces` works but won't result in each element having a valid index, instead its order in the sequence will be set.

new(verts, example=None)

Create a new edge from a given pair of verts.

PARAMETERS:

- **verts** (Sequence[BMVert]) – Vertex pair.
- **example** (BMEdge) – Existing edge to initialize settings (optional argument).

RETURNS:

The newly created edge.

RETURN TYPE:

BMEdge

remove(edge)

Remove an edge.

sort(key=None, reverse=False)

Sort the elements of this sequence, using an optional custom sort key. Indices of elements are not changed, `BMElemSeq.index_update` can be used for that.

PARAMETERS:

- **key** (Callable[[`BMVert` | `BMEdge` | `BMFace`], int] | None) – The key that sets the ordering of the elements.
- **reverse** (*bool*) – Reverse the order of the elements

Note

When the 'key' argument is not provided, the elements are reordered following their current index value. In particular this can be used by setting indices manually before calling this method.

Warning

Existing references to the N'th element, will continue to point the data at that index.

layers

custom-data layers (read-only).

TYPE:

`BMLayerAccessEdge`

class bmesh.types.**BMFaceSeq**

ensure_lookup_table()

Ensure internal data needed for int subscription is initialized with verts/edges/faces, eg `bm.verts[index]`.

This needs to be called again after adding/removing data in this sequence.

get(verts, fallback=None)

Return a face which uses the **verts** passed.

PARAMETERS:

- **verts** (Sequence[`BMVert`]) – Sequence of verts.
- **fallback** – Return this value if nothing is found.

RETURNS:

The face found or None

RETURN TYPE:

`BMFace`

index_update()

Initialize the index values of this sequence.

This is the equivalent of looping over all elements and assigning the index values.

```
for index, ele in enumerate(sequence):
    ele.index = index
```

Note

Running this on sequences besides `BMesh.verts`, `BMesh.edges`, `BMesh.faces` works but won't result in each element having a valid index, instead its order in the sequence will be set.

new(verts, example=None)

Create a new face from a given set of verts.

PARAMETERS:

- **verts** (Sequence[`BMVert`]) – Sequence of 3 or more verts.
- **example** (`BMFace`) – Existing face to initialize settings (optional argument).

RETURNS:

The newly created face.

RETURN TYPE:

`BMFace`

remove(face)

Remove a face.

sort(key=None, reverse=False)

Sort the elements of this sequence, using an optional custom sort key. Indices of elements are not changed, `BMElemSeq.index_update` can be used for that.

PARAMETERS:

- **key** (Callable[[`BMVert` | `BMEdge` | `BMFace`], int] | None) – The key that sets the ordering of the elements.
- **reverse** (*bool*) – Reverse the order of the elements

Note

When the 'key' argument is not provided, the elements are reordered following their current index value. In particular this can be used by setting indices manually before calling this method.

Warning

Existing references to the N'th element, will continue to point the data at that index.

active

active face.

TYPE:

`BMFace` or None

layers

custom-data layers (read-only).

TYPE:

`BMLayerAccessFace`

class bmesh.types.**BMLoopSeq**

layers

custom-data layers (read-only).

TYPE:

`BMLayerAccessLoop`

class bmesh.types.**BMIter**

Internal BMesh type for looping over verts/faces/edges, used for iterating over `BMElemSeq` types.

Selection History

class bmesh.types.**BMEditSelSeq**

add(element)

Add an element to the selection history (no action taken if its already added).

clear()

Empties the selection history.

discard(element)

Discard an element from the selection history.

Like remove but doesn't raise an error when the elements not in the selection list.

remove(element)

Remove an element from the selection history.

validate()

Ensures all elements in the selection history are selected.

active

The last selected element or None (read-only).

TYPE:

`BMVert`, `BMEdge` or `BMFace`

class bmesh.types.**BMEditSelIter**

Custom-Data Layer Access

class bmesh.types.**BMLayerAccessVert**

Exposes custom-data layer attributes.

bool

Generic boolean custom-data layer.

TYPE:

`BMLayerCollection` of boolean

color

Generic RGBA color with 8-bit precision custom-data layer.

TYPE:

`BMLayerCollection` of `mathutils.Vector`

deform

Vertex deform weight `BMDeformVert` (TODO).

TYPE:

`BMLayerCollection` of `bmesh.types.BMDeformVert`

float

Generic float custom-data layer.

TYPE:

`BMLayerCollection` of float

float_color

Generic RGBA color with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float_vector

Generic 3D vector with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

int

Generic int custom-data layer.

TYPE:

BMLayerCollection of int

shape

Vertex shapekey absolute location (as a 3D Vector).

TYPE:

BMLayerCollection of mathutils.Vector

skin

Accessor for skin layer.

TYPE:

BMLayerCollection of bmesh.types.BMVertSkin

string

Generic string custom-data layer (exposed as bytes, 255 max length).

TYPE:

BMLayerCollection of bytes

class bmesh.types.**BMLayerAccessEdge**

Exposes custom-data layer attributes.

bool

Generic boolean custom-data layer.

TYPE:

BMLayerCollection of boolean

color

Generic RGBA color with 8-bit precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float

Generic float custom-data layer.

TYPE:

BMLayerCollection of float

float_color

Generic RGBA color with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float_vector

Generic 3D vector with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

freestyle

Accessor for Freestyle edge layer.

TYPE:

BMLayerCollection

int

Generic int custom-data layer.

TYPE:

BMLayerCollection of int

string

Generic string custom-data layer (exposed as bytes, 255 max length).

TYPE:

BMLayerCollection of bytes

class bmesh.types.**BMLayerAccessFace**

Exposes custom-data layer attributes.

bool

Generic boolean custom-data layer.

TYPE:

BMLayerCollection of boolean

color

Generic RGBA color with 8-bit precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float

Generic float custom-data layer.

TYPE:

BMLayerCollection of float

float_color

Generic RGBA color with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float_vector

Generic 3D vector with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

freestyle

Accessor for Freestyle face layer.

TYPE:

BMLayerCollection

int

Generic int custom-data layer.

TYPE:

BMLayerCollection of int

string

Generic string custom-data layer (exposed as bytes, 255 max length).

TYPE:

BMLayerCollection of bytes

class bmesh.types.**BMLayerAccessLoop**

Exposes custom-data layer attributes.

bool

Generic boolean custom-data layer.

TYPE:

BMLayerCollection of boolean

color

Generic RGBA color with 8-bit precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float

Generic float custom-data layer.

TYPE:

BMLayerCollection of float

float_color

Generic RGBA color with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

float_vector

Generic 3D vector with float precision custom-data layer.

TYPE:

BMLayerCollection of mathutils.Vector

int

Generic int custom-data layer.

TYPE:

BMLayerCollection of int

string

Generic string custom-data layer (exposed as bytes, 255 max length).

TYPE:

BMLayerCollection of bytes

uv

Accessor for BMLoopUV UV (as a 2D Vector).

TYPE:

BMLayerCollection of bmesh.types.BMLoopUV

class bmesh.types.**BMLayerCollection**

Gives access to a collection of custom-data layers of the same type and behaves like Python dictionaries, except for the ability to do list like index access.

get(key, default=None)

Returns the value of the layer matching the key or default when not found (matches Python's dictionary function of the same name).

PARAMETERS:

- **key** (*str*) – The key associated with the layer.
- **default** (*Any*) – Optional argument for the value to return if *key* is not found.

items()

Return the identifiers of collection members (matching Python's dict.items() functionality).

RETURNS:

(key, value) pairs for each member of this collection.

RETURN TYPE:

list[tuple[str, BMLayerItem]]

keys()

Return the identifiers of collection members (matching Python's dict.keys() functionality).

RETURNS:

the identifiers for each member of this collection.

RETURN TYPE:

list[str]

new(name)

Create a new layer

PARAMETERS:

name (*str*) – Optional name argument (will be made unique).

RETURNS:

The newly created layer.

RETURN TYPE:

BMLayerItem

remove(layer)

Remove a layer

PARAMETERS:

layer (BMLayerItem) – The layer to remove.

values()

Return the values of collection (matching Python's dict.values() functionality).

RETURNS:

the members of this collection.

RETURN TYPE:

list[BMLayerItem]

verify()

Create a new layer or return an existing active layer

RETURNS:

The newly verified layer.

RETURN TYPE:

BMLayerItem

active

The active layer of this type (read-only).

TYPE:

BMLayerItem

is_singleton

True if there can exists only one layer of this type (read-only).

TYPE:

bool

class bmesh.types.**BMLayerItem**

Exposes a single custom data layer, their main purpose is for use as item accessors to custom-data when used with vert/edge/face/loop data.

copy_from(other)

Return a copy of the layer

PARAMETERS:

other (BMLayerItem) – Another layer to copy from.

name

The layers unique name (read-only).

TYPE:

str

Custom-Data Layer Types

class bmesh.types.**BMLoopUV**

pin_uv

UV pin state.

TYPE:

bool

select

UV select state.

TYPE:

bool

select_edge

UV edge select state.

TYPE:

bool

uv

Loops UV (as a 2D Vector).

TYPE:

mathutils.Vector

class bmesh.types.**BMDeformVert**

clear()

Clears all weights.

get(key, default=None)

Returns the deform weight matching the key or default when not found (matches Python's dictionary function of the same name).

PARAMETERS:

- **key** (*int*) – The key associated with deform weight.
- **default** (*Any*) – Optional argument for the value to return if *key* is not found.

items()

Return (group, weight) pairs for this vertex (matching Python's dict.items() functionality).

RETURNS:

(key, value) pairs for each deform weight of this vertex.

RETURN TYPE:

list[tuple[int, float]]

keys()

Return the group indices used by this vertex (matching Python's dict.keys() functionality).

RETURNS:

the deform group this vertex uses

RETURN TYPE:

list[int]

values()

Return the weights of the deform vertex (matching Python's dict.values() functionality).

RETURNS:

The weights that influence this vertex

RETURN TYPE:

list[float]