Skip to content

UIList(bpy_struct)

Basic UIList Example

This script is the UIList subclass used to show material slots, with a bunch of additional commentaries.

Notice the name of the class, this naming convention is similar as the one for panels or menus.

Note

UIList subclasses must be registered for blender to use them.

```
import bpy


class MATERIAL_UL_matslots_example(bpy.types.UIList):
    # The draw_item function is called for each item of the collection that is visible in
    #   data is the RNA object containing the collection,
    #   item is the current drawn item of the collection,
    #   icon is the "computed" icon for the item (as an integer, because some objects like
    #   have custom icons ID, which are not available as enum items).
    #   active_data is the RNA object containing the active property for the collection (i
    #   active item of the collection).
    #   active_propname is the name of the active property (use 'getattr(active_data, acti
    #   index is index of the current item in the collection.
    #   flt_flag is the result of the filtering process for this item.
    #   Note: as index and flt_flag are optional arguments, you do not have to use/declare
    #         need them.
    def draw_item(self, context, layout, data, item, icon, active_data, active_propname):
        ob = data
        slot = item
        ma = slot.material
        # draw_item must handle the three layout types... Usually 'DEFAULT' and 'COMPACT'
        if self.layout_type in {'DEFAULT', 'COMPACT'}:
            # You should always start your row layout by a label (icon + text), or a non-e
            # this will also make the row easily selectable in the list! The later also en
            # We use icon_value of label, as our given icon is an integer value, not an en
            # Note "data" names should never be translated!
            if ma:
                layout.prop(ma, "name", text="", emboss=False, icon_value=icon)
            else:
                layout.label(text="", translate=False, icon_value=icon)
        # 'GRID' layout type should be as compact as possible (typically a single icon!).
        elif self.layout_type == 'GRID':
            layout.alignment = 'CENTER'
            layout.label(text="", icon_value=icon)


# And now we can use this list everywhere in Blender. Here is a small example panel.
class UIListPanelExample1(bpy.types.Panel):
    """Creates a Panel in the Object properties window"""
    bl_label = "UIList Example 1 Panel"
    bl_idname = "OBJECT_PT_ui_list_example_1"
    bl_space_type = 'PROPERTIES'
    bl_space_type = 'PROPERTIES'
    bl_region_type = 'WINDOW'
    bl_context = "object"

    def draw(self, context):
        layout = self.layout

        obj = context.object

        # template_list now takes two new args.
        # The first one is the identifier of the registered UIList to use (if you want onl
        # with no custom draw code, use "UI_UL_list").
        layout.template_list("MATERIAL_UL_matslots_example", "", obj, "material_slots", ob

        # The second one can usually be left as an empty string.
        # It's an additional ID used to distinguish lists in case you use the same list se
        layout.template_list("MATERIAL_UL_matslots_example", "compact", obj, "material_slo
                             obj, "active_material_index", type='COMPACT')


def register():
    bpy.utils.register_class(MATERIAL_UL_matslots_example)
    bpy.utils.register_class(UIListPanelExample1)


def unregister():
    bpy.utils.unregister_class(UIListPanelExample1)
    bpy.utils.unregister_class(MATERIAL_UL_matslots_example)


if __name__ == "__main__":
    register()
```

Advanced UIList Example - Filtering and Reordering

This script is an extended version of the `UIList` subclass used to show vertex groups. It is not used ‘as is’, because iterating over all vertices in a ‘draw’ function is a very bad idea for UI performances! However, it’s a good example of how to create/use filtering/reordering callbacks.

```
import bpy


class MESH_UL_vgroups_slow(bpy.types.UIList):
    # Constants (flags)
    # Be careful not to shadow FILTER_ITEM!
    VGROUP_EMPTY = 1 << 0

    # Custom properties, saved with .blend file.
    use_filter_empty: bpy.props.BoolProperty(
        name="Filter Empty",
        default=False,
        options=set(),
        description="Whether to filter empty vertex groups",
    )
    use_filter_empty_reverse: bpy.props.BoolProperty(
        name="Reverse Empty",
        default=False,
        options=set(),
        description="Reverse empty filtering",
    )
    use_filter_name_reverse: bpy.props.BoolProperty(
        name="Reverse Name",
        default=False,
        options=set(),
        description="Reverse name filtering",
    )

    # This allows us to have mutually exclusive options, which are also all disable-able!
    def _gen_order_update(name1, name2):
        def _u(self, ctxt):
            if (getattr(self, name1)):
                setattr(self, name2, False)
        return _u
    use_order_name: bpy.props.BoolProperty(
        name="Name", default=False, options=set(),
        description="Sort groups by their name (case-insensitive)",
        update=_gen_order_update("use_order_name", "use_order_importance"),
    )
    use_order_importance: bpy.props.BoolProperty(
        name="Importance",
        default=False,
        options=set(),
        description="Sort groups by their average weight in the mesh",
        update=_gen_order_update("use_order_importance", "use_order_name"),
    )

    # Usual draw item function.
    def draw_item(self, context, layout, data, item, icon, active_data, active_propname, i
        # Just in case, we do not use it here!
        self.use_filter_invert = False

        # assert(isinstance(item, bpy.types.VertexGroup)
        vgroup = item
        if self.layout_type in {'DEFAULT', 'COMPACT'}:
            # Here we use one feature of new filtering feature: it can pass data to draw_i
            # parameter, which contains exactly what filter_items set in its filter list f
            # In this case, we show empty groups grayed out.
            if flt_flag & self.VGROUP_EMPTY:
                col = layout.column()
                col.enabled = False
                col.alignment = 'LEFT'
                col.prop(vgroup, "name", text="", emboss=False, icon_value=icon)
            else:
                layout.prop(vgroup, "name", text="", emboss=False, icon_value=icon)
            icon = 'LOCKED' if vgroup.lock_weight else 'UNLOCKED'
            layout.prop(vgroup, "lock_weight", text="", icon=icon, emboss=False)
        elif self.layout_type == 'GRID':
            layout.alignment = 'CENTER'
            if flt_flag & self.VGROUP_EMPTY:
                layout.enabled = False
            layout.label(text="", icon_value=icon)

    def draw_filter(self, context, layout):
        # Nothing much to say here, it's usual UI code...
        row = layout.row()

        subrow = row.row(align=True)
        subrow.prop(self, "filter_name", text="")
        icon = 'ZOOM_OUT' if self.use_filter_name_reverse else 'ZOOM_IN'
        subrow.prop(self, "use_filter_name_reverse", text="", icon=icon)

        subrow = row.row(align=True)
        subrow.prop(self, "use_filter_empty", toggle=True)
        icon = 'ZOOM_OUT' if self.use_filter_empty_reverse else 'ZOOM_IN'
        subrow.prop(self, "use_filter_empty_reverse", text="", icon=icon)

        row = layout.row(align=True)
        row.label(text="Order by:")
        row.prop(self, "use_order_name", toggle=True)
        row.prop(self, "use_order_importance", toggle=True)
        icon = 'TRIA_UP' if self.use_filter_orderby_invert else 'TRIA_DOWN'
        row.prop(self, "use_filter_orderby_invert", text="", icon=icon)

    def filter_items_empty_vgroups(self, context, vgroups):
        # This helper function checks vgroups to find out whether they are empty, and what
        # TODO: This should be RNA helper actually (a vgroup prop like "raw_data: ((vidx,
        #       Too slow for python!
        obj_data = context.active_object.data
        ret = {vg.index: [True, 0.0] for vg in vgroups}
        if hasattr(obj_data, "vertices"):  # Mesh data
            if obj_data.is_editmode:
                import bmesh
                bm = bmesh.from_edit_mesh(obj_data)
                # only ever one deform weight layer
                dvert_lay = bm.verts.layers.deform.active
                fact = 1 / len(bm.verts)
                if dvert_lay:
                    for v in bm.verts:
                        for vg_idx, vg_weight in v[dvert_lay].items():
                            ret[vg_idx][0] = False
                            ret[vg_idx][1] += vg_weight * fact
            else:
                fact = 1 / len(obj_data.vertices)
                for v in obj_data.vertices:
                    for vg in v.groups:
                        ret[vg.group][0] = False
                        ret[vg.group][1] += vg.weight * fact
        elif hasattr(obj_data, "points"):  # Lattice data
            # XXX no access to lattice editdata?
            fact = 1 / len(obj_data.points)
            for v in obj_data.points:
                for vg in v.groups:
                    ret[vg.group][0] = False
                    ret[vg.group][1] += vg.weight * fact
        return ret
        return ret

    def filter_items(self, context, data, propname):
        # This function gets the collection property (as the usual tuple (data, propname))
        # * The first one is for filtering, it must contain 32bit integers were self.bitfl
        #   matching item as filtered (i.e. to be shown). The upper 16 bits (including sel
        #   reserved for internal use, the lower 16 bits are free for custom use. Here we
        #   VGROUP_EMPTY.
        # * The second one is for reordering, it must return a list containing the new inc
        #   gives us a mapping org_idx -> new_idx).
        # Please note that the default UI_UL_list defines helper functions for common task
        # If you do not make filtering and/or ordering, return empty list(s) (this will be
        # returning full lists doing nothing!).
        vgroups = getattr(data, propname)
        helper_funcs = bpy.types.UI_UL_list

        # Default return values.
        flt_flags = []
        flt_neworder = []

        # Pre-compute of vgroups data, CPU-intensive. :/
        vgroups_empty = self.filter_items_empty_vgroups(context, vgroups)

        # Filtering by name
        if self.filter_name:
            flt_flags = helper_funcs.filter_items_by_name(self.filter_name, self.bitflag_f
                                                          reverse=self.use_filter_name_rev
        if not flt_flags:
            flt_flags = [self.bitflag_filter_item] * len(vgroups)

        # Filter by emptiness.
        for idx, vg in enumerate(vgroups):
            if vgroups_empty[vg.index][0]:
                flt_flags[idx] |= self.VGROUP_EMPTY
                if self.use_filter_empty and self.use_filter_empty_reverse:
                    flt_flags[idx] &= ~self.bitflag_filter_item
            elif self.use_filter_empty and not self.use_filter_empty_reverse:
                flt_flags[idx] &= ~self.bitflag_filter_item

        # Reorder by name or average weight.
        if self.use_order_name:
            flt_neworder = helper_funcs.sort_items_by_name(vgroups, "name")
        elif self.use_order_importance:
            _sort = [(idx, vgroups_empty[vg.index][1]) for idx, vg in enumerate(vgroups)]
            flt_neworder = helper_funcs.sort_items_helper(_sort, lambda e: e[1], True)

        return flt_flags, flt_neworder


# Minimal code to use above UIList...
class UIListPanelExample2(bpy.types.Panel):
    """Creates a Panel in the Object properties window"""
    bl_label = "UIList Example 2 Panel"
    bl_idname = "OBJECT_PT_ui_list_example_2"
    bl_space_type = 'PROPERTIES'
    bl_region_type = 'WINDOW'
    bl_context = "object"

    def draw(self, context):
        layout = self.layout
        obj = context.object

        # template_list now takes two new args.
        # The first one is the identifier of the registered UIList to use (if you want onl
        # with no custom draw code, use "UI_UL_list").
        layout.template_list("MESH_UL_vgroups_slow", "", obj, "vertex_groups", obj.vertex_


def register():
    bpy.utils.register_class(MESH_UL_vgroups_slow)
    bpy.utils.register_class(UIListPanelExample2)


def unregister():
    bpy.utils.unregister_class(UIListPanelExample2)
    bpy.utils.unregister_class(MESH_UL_vgroups_slow)


if __name__ == "__main__":
    register()
```

base class — `bpy_struct`

subclasses — `ASSETBROWSER_UL_metadata_tags`, `CLIP_UL_tracking_objects`, `CURVES_UL_attributes`, `DATA_UL_bone_collections`, `FILEBROWSER_UL_dir`, `GPENCIL_UL_annotation_layer`, `GPENCIL_UL_layer`, `GPENCIL_UL_masks`, `GPENCIL_UL_matslots`, `GREASE_PENCIL_UL_attributes`, `GREASE_PENCIL_UL_masks`, `IMAGE_UL_render_slots`, `IMAGE_UL_udim_tiles`, `MASK_UL_layers`, `MATERIAL_UL_matslots`, `MESH_UL_attributes`, `MESH_UL_color_attributes`, `MESH_UL_color_attributes_selector`, `MESH_UL_shape_keys`, `MESH_UL_uvmaps`, `MESH_UL_vgroups`, `PARTICLE_UL_particle_systems`, `PHYSICS_UL_dynapaint_surfaces`, `POINTCLOUD_UL_attributes`, `POSE_UL_selection_set`, `RENDER_UL_renderviews`, `SCENE_UL_gltf2_filter_action`, `SCENE_UL_keying_set_paths`, `TEXTURE_UL_texpaintslots`, `TEXTURE_UL_texslots`, `UI_UL_list`, `USERPREF_UL_asset_libraries`, `USERPREF_UL_extension_repos`, `VIEWLAYER_UL_aov`, `VIEWLAYER_UL_linesets`, `VOLUME_UL_grids`, `WORKSPACE_UL_addons_items`

class bpy.types.**UIList(bpy_struct)**

UI list containing the elements of a collection

bitflag_filter_item

The value of the reserved bitflag ‘FILTER_ITEM’ (in filter_flags values)

TYPE:

int in [0, inf], default 0, (readonly)

bl_idname

If this is set, the uilist gets a custom ID, otherwise it takes the name of the class used to define the uilist (for example, if the class name is “OBJECT_UL_vgroups”, and bl_idname is not set by the script, then bl_idname = “OBJECT_UL_vgroups”)

TYPE:

string, default “”, (never None)

filter_name

Only show items matching this name (use '*' as wildcard)

TYPE:

string, default "", (never None)

layout_type

TYPE:

enum in Uilist Layout Type Items, default 'DEFAULT', (readonly)

list_id

Identifier of the list, if any was passed to the "list_id" parameter of "template_list()"

TYPE:

string, default "", (readonly, never None)

use_filter_invert

Invert filtering (show hidden items, and vice versa)

TYPE:

boolean, default False

use_filter_show

Show filtering options

TYPE:

boolean, default False

use_filter_sort_alpha

Sort items by their name

TYPE:

boolean, default False

use_filter_sort_lock

Lock the order of shown items (user cannot change it)

TYPE:

boolean, default False

use_filter_sort_reverse

Reverse the order of shown items

TYPE:

boolean, default False

draw_item(context, layout, data, item, icon, active_data, active_property, index, flt_flag)

Draw an item in the list (NOTE: when you define your own draw_item function, you may want to check given 'item' is of the right type…)

PARAMETERS:

- **layout** (`UILayout` , (never None)) – Layout to draw the item
- **data** (`AnyType`) – Data from which to take Collection property
- **item** (`AnyType`) – Item of the collection property
- **icon** (*int in [0, inf]*) – Icon of the item in the collection
- **active_data** (`AnyType` , (never None)) – Data from which to take property for the active element
- **active_property** (*string, (optional argument, never None)*) – Identifier of property in active_data, for the active element
- **index** (*int in [0, inf]*) – Index of the item in the collection

- **flt_flag** (*int in [0, inf]*) – The filter-flag result for this item

draw_filter(context, layout)

Draw filtering options

PARAMETERS:

layout (`UILayout`, (never None)) – Layout to draw the item

filter_items(context, data, property)

Filter and/or re-order items of the collection (output filter results in filter_flags, and reorder results in filter_neworder arrays)

PARAMETERS:

- **data** (`AnyType`) – Data from which to take Collection property
- **property** (*string, (never None)*) – Identifier of property in data, for the collection

RETURNS:

filter_flags, An array of filter flags, one for each item in the collection (NOTE: The upper 16 bits, including FILTER_ITEM, are reserve only use the lower 16 bits for custom usages), int array of 1 items in [0, inf]

filter_neworder, An array of indices, one for each item in the collection, mapping the org index to the new one, int array of 1 items in [0 inf]

RETURN TYPE:

(int array of 1 items in [0, inf], int array of 1 items in [0, inf])

classmethod append(draw_func)

Append a draw function to this menu, takes the same arguments as the menus draw function

classmethod is_extended()

classmethod prepend(draw_func)

Prepend a draw function to this menu, takes the same arguments as the menus draw function

classmethod remove(draw_func)

Remove a draw function that has been added to this menu

classmethod bl_rna_get_subclass(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The RNA type or default when not found.

RETURN TYPE:

`bpy.types.Struct` subclass

classmethod bl_rna_get_subclass_py(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The class or default when not found.

RETURN TYPE:

type

Inherited Properties

- `bpy_struct.id_data`

Inherited Functions

- `bpy_struct.as_pointer`
- `bpy_struct.driver_add`
- `bpy_struct.driver_remove`
- `bpy_struct.get`
- `bpy_struct.id_properties_clear`
- `bpy_struct.id_properties_ensure`
- `bpy_struct.id_properties_ui`
- `bpy_struct.is_property_hidden`
- `bpy_struct.is_property_overridable_library`
- `bpy_struct.is_property_readonly`
- `bpy_struct.is_property_set`
- `bpy_struct.items`
- `bpy_struct.keyframe_delete`
- `bpy_struct.keyframe_insert`
- `bpy_struct.keys`
- `bpy_struct.path_from_id`
- `bpy_struct.path_resolve`
- `bpy_struct.pop`
- `bpy_struct.property_overridable_library_set`
- `bpy_struct.property_unset`
- `bpy_struct.type_recast`
- `bpy_struct.values`