Skip to content

Bones & Armatures

Edit Bones, Pose Bones, Bone… Bones

Armature Bones in Blender have three distinct data structures that contain them. If you are accessing the bones through one of them, you may not have access to the properties you really need.

> Note
>
> In the following examples `bpy.context.object` is assumed to be an armature object.

Edit Bones

`bpy.context.object.data.edit_bones` contains an edit bones; to access them you must set the armature mode to Edit-Mode first (ed bones do not exist in Object or Pose-Mode). Use these to create new bones, set their head/tail or roll, change their parenting relationships to other bone etc.

Example using `bpy.types.EditBone` in armature Edit-Mode which is only possible in Edit-Mode:

```
>>> bpy.context.object.data.edit_bones["Bone"].head = Vector((1.0, 2.0, 3.0))
```

This will be empty outside of Edit-Mode:

```
>>> mybones = bpy.context.selected_editable_bones
```

Returns an edit bone only in Edit-Mode:

```
>>> bpy.context.active_bone
```

Bones (Object-Mode)

`bpy.context.object.data.bones` contains bones. These *live* in Object-Mode, and have various properties you can change, note that the head and tail properties are read-only.

Example using `bpy.types.Bone` in Object or Pose-Mode returning a bone (not an edit bone) outside of Edit-Mode:

```
>>> bpy.context.active_bone
```

This works, as with Blender the setting can be edited in any mode:

```
>>> bpy.context.object.data.bones["Bone"].use_deform = True
```

Accessible but read-only:

```
>>> tail = myobj.data.bones["Bone"].tail
```

Pose Bones

`bpy.context.object.pose.bones` contains pose bones. This is where animation data resides, i.e. animatable transformations are applied pose bones, as are constraints and IK-settings.

Examples using `bpy.types.PoseBone` in Object or Pose-Mode:

```
# Gets the name of the first constraint (if it exists)
bpy.context.object.pose.bones["Bone"].constraints[0].name
```

```
bpy.context.object.pose.bones["Bone"].constraints[0].name

# Gets the last selected pose bone (Pose-Mode only)
bpy.context.active_pose_bone
```

Note

Notice the pose is accessed from the object rather than the object data, this is why Blender can have two or more objects sharing the same armature in different poses.

Note

Strictly speaking pose bones are not bones, they are just the state of the armature, stored in the `bpy.types.Object` rather than the `bpy.types.Armature`, yet the real bones are accessible from the pose bones via `bpy.types.PoseBone.bone`.

Armature Mode Switching

While writing scripts that deal with armatures you may find you have to switch between modes, when doing so take care when switching out of Edit-Mo not to keep references to the edit bones or their head/tail vectors. Further access to these will crash Blender so it's important that the script clearly separates sections of the code which operate in different modes.

This is mainly an issue with Edit-Mode since pose data can be manipulated without having to be in Pose-Mode, yet for operator access you may still nee to enter Pose-Mode.