EXTENSION

### Attributes vs Properties

| **Term** | **What it is** | **Example** |
| --- | --- | --- |
| **Attribute** | A regular Python variable attached to an object. You can freely read/write it. | self.baseColorList, self.ownerComp |
| **Property** | A special variable created using TDF.createProperty() that is more integrated into TouchDesigner (e.g. it can be shown in the parameter window and marked as dependable). |  |

**Attributes** are like internal variables.

**Properties** are "exposed" variables that can be tracked/reactive or read-only.

python

CopyEdit

TDF.createProperty(self, 'BaseColor',

value=self.baseColorList[self.ColorIndex],

readOnly=True,

dependable=True)

### ****two things**** on your extension object (self):

| **Name** | **What it is** | **Example Use** |
| --- | --- | --- |
| self.BaseColor | The actual value of the property | print(self.BaseColor) gives [1, 0, 0] |
| self.\_BaseColor | A special internal object (the property wrapper) | self.\_BaseColor.val = [0, 1, 0] updates the value |

TouchDesigner separates the **value** from the **controller** of that value.

Think of it like this:

| **What** | **Analogy** |
| --- | --- |
| self.BaseColor | Like reading the color of a lightbulb |
| self.\_BaseColor | Like the light switch or dimmer that controls the bulb |

Kinda… but not really.

| **Concept** | **Python "private"** | **TouchDesigner "not promoted"** |
| --- | --- | --- |
| Access control | Just a naming hint (like \_myVar) | Actually hides from component UI |
| Enforcement | Not enforced (can still access it) | Hidden unless you go through .ext.Extension |
| Purpose | Signal to devs not to use it | Clean up UI and only expose what’s needed |

So, it’s **similar** in that:

**Non-promoted** attributes are more “internal” (like \_myVar)

**Promoted** ones are meant to be used outside the component (like public methods)

…but it’s a UI & accessibility thing, **not enforced privacy**.

| **Term** | **Meaning** |
| --- | --- |
| **Promoted Attribute** | A capitalized property or method (self.MyVar) that becomes accessible from op('comp').MyVar if the extension is promoted. |
| **Non-Promoted** | A lowercase attribute (e.g., self.myVar) that is only available through op('comp').ext.MyExtension.myVar. |
| **Not the same as "private"** | Promotion is about UI exposure, not code access restrictions. |

### What is .ext?

.ext is **TouchDesigner’s special reference** to a component's **extension object**. It’s where the component's **custom Python class** (your ScatterplotGenerator, for example) is attached.

When you define an extension class like this:

python

CopyEdit

class ScatterplotGenerator:

def \_\_init\_\_(self, ownerComp):

self.ownerComp = ownerComp

self.B = 1 # promoted attribute

The extension is **not automatically available** unless it's promoted through the **"Promote Extension"** parameter. If it is promoted, it essentially **exposes the entire extension class** as an accessible object within the component.

### So, ****What is Happening**** when you do op('scatterplot').ext?

op('scatterplot') is **Accessing the Component**:

op('scatterplot') refers to the **TouchDesigner operator** (the component) called scatterplot.

op('scatterplot').ext is **Accessing the Extension Object**:

.ext is a reference to the **extension** (which is your ScatterplotGenerator class), **attached** to the component.

This is where the **logic** and **attributes** of your custom extension are stored.

So op('scatterplot').ext is just **TouchDesigner’s way of exposing the Python class** tied to that component. It’s **not a standard Python object** but a **special object** created by TouchDesigner for easier access to the extension class.

op('scatterplot').ext gives you access to the **extension object** — this is where all the **custom Python logic** and **data** reside.

**Promoted attributes** are automatically accessible **directly** from op('scatterplot') (with the capitalized name).

**Non-promoted attributes** can only be accessed through .ext.

### Extension = Python Class for Custom Behavior

An **extension** is essentially a **Python class** attached to a component in TouchDesigner.

It is used to add custom functionality to the component, often by creating new properties, methods, or behaviors that are not natively available through the TouchDesigner interface.

### How It Works:

**Extension Class Definition:**

You define an extension class in Python, which contains **attributes** (data) and **methods** (functions).

This class is then attached to a **component** using the .ext attribute, which makes the extension available to that component.

**Promoting Extension:**

If you **promote** an extension, its **attributes and methods** can be accessed **externally** using the component’s interface, just like any other parameter.

A promoted extension allows you to call methods or set/get attributes directly from outside the component (e.g., op('yourComponent').PromotedFunction()).

so also method can be promoted or not

###### ChatGPT said:

Yes, methods can be promoted in TouchDesigner as well, though there are some distinctions compared to attributes.

### Here's the difference:

**Promoted Attributes**:

Attributes (like a and B in your example) can be promoted through the component's UI, which makes them accessible directly from the component (like op('yourComp').B).

Promoting an attribute allows you to use it as a parameter in the component's parameter window.

**Promoted Methods**:

Similarly, methods can be promoted, but in this case, you need to follow a specific process:

When you promote a method, you typically make it available as a callable function directly from the component.

This can be done by setting the Promote parameter for that method in the component’s UI.

Once promoted, you can call the method as if it were a parameter of the component.

- WHAT IS AN EXTENSION

- EXTENSIONS AND PROMOTING ATTRIBUTES AND METHODS

- DIFFERENCE ATTRIBUTE PROPERTY