COURSE

1. CREATING AN EXTENSION

Material

<https://derivative.ca/UserGuide/Extensions#Promoting_Extensions>

<https://matthewragan.com/2015/07/05/touchdesigner-understanding-extensions/>

You're working with a **TouchDesigner extension**, which is basically a Python class that adds custom behavior and data to a component (COMP).

| **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

| **What** | **Meaning** |
| --- | --- |
| Dependency | A value that triggers updates when it changes |
| .val | The actual stored value of the property |
| dependable=True | Enables this behavior when creating the property |

PRACTIVE

extension = a custom class attached to a component that expand its behaviour

component BASE op("scatterplot").ext

ext --> a way of exposing python class tied to the component, it's a reference to the extension

class ScatterplotGenerator:

...

op("scatterplot").ext.ScatterplotGenerator

op("scatterplot") --> finds the coponent

.ext --> points to the extension (class ScatterplotGenerator)

.ScatterplotGenerator --> access the class itself, so then I can access its properties

a, B: attributes,

debut, PromotedDebug: methods

Methods and attributes can be promoted or not

TO ACCESS:

ext = op('scatterplot')

print(ext.B)

ext.PromotedDebug()

ext = op('scatterplot').ext.ScatterplotGenerator

print(ext.a)

ext.debug()

PYTHON PROPERTIES

In Python, **properties** are special kinds of functions (typically getter/setter methods) that you **access like attributes**, but behind the scenes, they run code.

PROPERTY IN TD

In TouchDesigner, using TDF.createProperty(...) gives you:

#### ✅ 1. ****Attribute-like access****

print(self.ScatterPlotGenerator)

self.ScatterPlotGenerator = 10

But behind the scenes, this is **actually managing data via a special property object**.

#### ✅ 2. ****Dependable behavior****

If dependable=True, this property can be **used in DAT expressions**, and TD knows that when this value changes, dependent operators might need to update.

That’s useful for **reactive networks**, where one thing should automatically respond to a change in another.

#### ✅ 3. ****Optional read-only protection****

With readOnly=True, it becomes a value that can be read, but not changed externally — kind of like a read-only property in Python.

### dependable=True?

Set dependable=True if you want TouchDesigner to automatically trigger updates when the value changes (like cooking a DAT or CHOP, or triggering an event).

### ✅ Why parameter\_execute works well with extensions:

Using a parameter\_execute DAT allows you to **separate UI logic from functionality**. Here's how it fits together cleanly:

**Parameters** live on your component and can be UI-facing (e.g. drop-downs, pulses, toggles).

**parameter\_execute DAT** watches for changes to those parameters.

When a parameter changes, parameter\_execute **triggers the relevant method** in your Extension class.

Your **Extension** holds all the logic, state (with StorageManager), and methods.