Anatomy of a custom Dplug widget

How do widget work?
Which callbacks to answer?
When to call setDirtyWhole? etc.
Plug-in UIs are important

- It highlights the unique features of your plug-in. 
  *If the UI doesn't say loudly that a feature is interesting, then it will be considered not-new.*

- Custom widgets brings your UX desire to reality.  
  *“I have a dream. I want to set both Ratio and Threshold at the same time”*

- Dplug allows Scriptable + Resizeable + PBR UI

  *2022 limitations: not DPI aware on Mac, no dynamic widget creation.*
Let's build that widget: a stereo width control

When dragged

It takes 183 LOC.

When mouse is over

Rest position
Let's build that widget: a stereo width control

NOT THAT TRIVIAL TO CODE.

When dragged

It takes 183 LOC.

When mouse is over

Rest position
Let’s go over those 183 lines, one by one.
You should need to import that class name from Wren. The name don’t _have_ to start with UI, I like to do that for UIElement derivatives.
Expose what you want for Wren to see.

Tip: if you draw something complicated, use temporary @ScriptProperty that are not “styling” but content. (eg: control points in a curve)
Register and unregister the widget as a Listener for each parameter that will change the graphics when changing.

Stereo width parameter

The “section enable” BoolParameter

DO NOT FORGET TO BALANCE CALLS:
- addListener
- removeListener
WHAT THE UI ELEMENT FLAGS MEAN

flagRaw ⇔ onDrawRaw is called.
This widget can write stuff on the Raw level, always on top of the PBR level. Suitable for 60FPS.

flagAnimated ⇔ onAnimate is called
Typically used to request a redraw if things have changed.

flagPBR ⇔ onDrawPBR is called
This widget can write stuff on the PBR level.
This is not suitable for 60FPS display, so this sort of widget has to call setDirty infrequently.
DRAWING PART 1

Raw texture, cropped to the widget position. Pixel (0, 0) is top-left of widget.

You can modify only inside the dirtyRects. Their position are relative to rawMap (local coordinates).

Get widget size by reading position(). This is the “world” position, not the position in rawMap.

Only worthwhile for width and height.

Repeat cropped drawing for each dirtyRect since it’s generally 1 or 2 rects.

override void onDrawRaw(ImageRef!RGBA rawMap, box2i[] dirtyRects)
{
    bool enabled = _enableParam.value();

    float W = position.width;
    float H = position.height;

    float center = W * 0.5f;
    float extent = W * 0.49f * 0.8f; // more esthetic to leave a bit of secondary color
    float baseExtent = W * 0.03f; // so that a line does appear for 0% width

    float width = _param.getNormalized(); // 0 to 1

    foreach(dirtyRect; dirtyRects)
    {
        auto cRaw = rawMap.cropImageRef(dirtyRect);
        canvas.initialize(cRaw);
        canvas.translate(-dirtyRect.min.x, -dirtyRect.min.y);
    }

    translate() call => so that you can draw in local coordinates.
Pro-tip = Have key metrics in the widget depend on the widget size (width and height).

This allows to:

1. Have less work to do in `reflow()` (eg: `fontSizePx`)

2. Resize things interactively with right-click in debug mode.

3. Easier to scale UI.

>>> AVOID PIXEL QUANTITIES IF YOU CAN! <<<
The drawing itself.

Using `@ScriptProperty` values instead of hardcoded values will make the widget more reusable.
LISTENING TO PARAMETER CHANGES

Important: A widget is **not** redrawn unless you call `setDirty[Whole]`.

Called when the parameter is changed by UI interaction **OR** host automation.

=> *call setDirtyWhole since DAW automation wouldn't redraw else.*

Called when the UI calls `beginParamEdit` / `endParamEdit()`.

=> *we call setDirtyWhole there too, unless you do it* when you start dragging, or stop dragging. Perhaps not necessary in that widget.

*Most often this can be left empty because you will have onMouseClick / onBeginDrag / onStopDrag...*
You can optimize rendering of a large widget by redrawing only the portion of its position rectangle, that has changed.

**setDirty**  
Redraws a single rectangle area, given in local coordinates. And the widgets beneath it.

**setDirtyWhole()**  
Redraws the whole position rectangle, and the widgets beneath it.

*Tip: Partial rectangles are generated anyway when another window pass above your plug-in, so you have to handle it anyway. And that's why dirtyRects exists.*
RESPONDING TO A MOUSE CLICK

>>> on Mouse Click <<<

Parameter reset code. Can't set parameters outside of a balanced `beginParamEdit/endParamEdit` pair.

```cpp
override bool onMouseClick(int x, int y, int button, bool isDoubleClick, MouseState mstate)
{
    // double-click => set to default
    if (isDoubleClick || mstate.altPressed)
    {
        _param.beginParamEdit();
        _param.setFromGUI(_param.defaultValue());
        _param.endParamEdit();
    }
    return true; // to initiate dragging
}
```

VERY IMPORTANT

Returning true = Start a drag operation. Window captures mouse until release.

If you don't need to do anything while mouse dragging, you still have to return true to consider the click handled.

Returning false = Consider the click unhandled. The event passed down to children etc.
A mouse drag happens whenever `onMouseClick` returned `true`.

Dragging has a start and an ending, it can be used to call balanced pairs of `beginParamEdit()`/`endParamEdit()`.

```java
override void onBeginDrag()
{
    _param.beginParamEdit();
}

override void onStopDrag()
{
    _param.endParamEdit();
}
```

**Tip:** eventually use a state machine to know which parameter you are dragging, it is useful for controls that have several points to drag.
RESPONDING TO A MOUSE DRAG Part 2

Slider logic like UISlider

Uses mouse displacement dx and dy (not as precise to use x and y, but can be done)

Divide by height to be as sensitive at every plugin size.

WHO IS CALLED?
- within dragging: => onMouseDrag
- without dragging => onMouseMove

Using dx/dy needs to read current parameter value.

```java
@Override
void onMouseDrag(int x, int y, int dx, int dy, MouseState mstate) {

    // FUTURE: replace by actual trail height instead of total height
    float displacementInHeight = cast(float)(dy) / _position.height;

    float modifier = 1.0f;
    if (mstate.shiftPressed || mstate.ctrlPressed) modifier *= 0.1f;

    double oldParamValue = _param.getNormalized();
    double newParamValue = oldParamValue - displacementInHeight * modifier * sensitivity;
    if (mstate.altPressed) newParamValue = _param.getDefaultValue();

    if (y > _mousePosOnLast0Cross) return;
    if (y < _mousePosOnLast1Cross) return;
    if (newParamValue <= 0 && oldParamValue > 0) _mousePosOnLast0Cross = y;
    if (newParamValue >= 1 && oldParamValue < 1) _mousePosOnLast1Cross = y;
    if (newParamValue < 0) newParamValue = 0;
    if (newParamValue > 1) newParamValue = 1;
    if (newParamValue > 0) _mousePosOnLast0Cross = float.infinite;
    if (newParamValue < 1) _mousePosOnLast1Cross = -float.infinite;

    if (newParamValue != oldParamValue) {
        if (auto p = cast(FloatParameter)_param) {
            p.setFromGUINormalized(newParamValue);
        } else {
            assert(false); // only float parameters supported
        }
    }
}
```
Called when mouse enters the widget. Here we trigger redraw because it may trigger selection highlight.

```java
override void onMouseEnter()
{
    setDirtyWhole();
}
```

Also called here because it may lose selection highlight.

**onMouseMove may not be generated when the mouse exits the widget!**

=> Use **onMouseExit** if you have a visual change just by hovering the mouse.
Must hold its own Canvas, because several widget can be drawn simultaneously if they don't intersect. **Don't share Canvas instances.**

Hold parameters it reads/write values from

Slider logic taken from UISlider

```cpp
private:
    Canvas canvas;
    FloatParameter _param;
    BoolParameter _enableParam;
    float _mousePosOnLast0Cross;
    float _mousePosOnLast1Cross;

    void clearCrosspoints()
    {
        _mousePosOnLast0Cross = float.infinity;
        _mousePosOnLast1Cross = -float.infinity;
    }
```
FUTURE

- Documenting all that a bit more.

- 3 possible return values for `onMouseClick`
  - Event consumed, start Dragging.
  - Event consumed, do not start Dragging.
  - Event not consumed.
Questions?