UnderPressure
by Peter Graffagnino

UnderPressure is a simple application demonstrating how to take advantage of the pressure sensitive features of graphics tablets under NeXTSTEP 3.0. It is basically a simple paint program that draws variable-width brush strokes. The application consists of only two custom classes **PressureView** and **Brush**. **PressureView** serves two illustrative purposes. Most importantly, it shows how to set up a view to inform the event system that it is interested in pressure events. Secondarily, it shows how to implement a View with a backing store using **NXImage**. The **Brush** class implements a simple moveto/lineto brush protocol, and demonstrates a reasonably efficient way to draw a variable width line using linecaps and trapezoidal fillets.

Areas of Interest Demonstrated in UnderPressure

· maintaing special event trackingrects for pressure and event coalescing (PressureView)
· a simple paint-brush model (Brush)
· fast filled circles using linecaps (Brush)
· using an NXImage to provide a backing store for a View (PressureView)
· using phantom outlets to set default control values at nib loading (Brush)
· handling pen proximity events as flags changed (PressureView)

Technical Details of Pressure Support in 3.0.

NeXTSTEP 3.0 supports the pressure-sensitive features of WACOM tablets. The pressure-pen emits a continuous range of values from its pressure-sensitive tip. In order for the pen to be used as a generic pointing device (i.e. a mouse substitute), mouse-up and mouse-down events need to be synthesized by the low-level tablet driver. This is done using a pair of thresholds to implement a "debounced" binary switch -- a fairly large pressure threshold is used to determine when to switch from mouse-up to mouse-down, and a fairly light pressure threshold is used to determine when to switch from mouse-down to mouse-up. This behavior enables the pressure pen to be used with any application -- emulating a mouse as closely as possible. However, for applications that want to take advantage of pressure, this is clearly not the right model. For example, if a paint program goes into its modal painting loop on mouseDown:, it would loose all of the valuable pressure data which was lighter than the mouseDown threshold! To solve this

problem, applications that are pressure-aware need to inform the event system that they are interested in all pressure events (not just mouse emulation). In this mode the system will treat any non-zero pressure as a mouse-down, and zero pressure as mouse-up. Since this is most useful on a View basis, this mode is set up using the array form of WindowServer's **settrackingrect** operator, which is new in 3.0 (see **pressurerect.psw**). As **PressureView** demonstrates, it's fairly easy to maintain the appropriate tracking rect, by overriding the **resetCursorRects** method in View. Notice that the extra sensitivity in **UnderPressure** only takes place in the **PressureView**, not on the other controls in the window. -- a light stroke on the canvas will mark a thin line, whereas a similar stroke on the scrollbar will have no effect. This is because outside the tracking rectangle set up by **PressureView**, normal mouse emulation thresholds are used to determine mouseDown.

Another feature added to the event subsystem for 3.0 is the ability to disable event coalescing in the system. Normally in NeXTSTEP events are aggressively coalesced making the feedback loop as tight as possible. Sometimes, particularly with pen input devices, this is not what an application might want. For example, an application digitizing a pen-stroke for handwriting recognition, might want all of the data from the input device. Also, a painting program might in fact want to lag somewhat, in order to more faithfully render the user's gesture. In 3.0, event-coalescing can also be controlled on a per-trackingrect basis. **PressureView** also demonstrates this.

Event coalescing occurs at two different levels in NeXTSTEP. All events flow from the input device, to the kernel, from the kernel to the WindowServer, and from the WindowServer to dpsclient (in your application). Events can get coaleseced in the kernel or in dpsclient. When a tracking rect is setup to disable event coalescing, the kernel will avoid coalescing the event, and stamp in with the **NX_NONCOALSCEDFLAG** in the event flag field. As this event percolates up through the system, other layers of software will avoid coalecing the event. (Note: dpsclient event coalecing can be controlled separately with the **DPSSetTracking**() function, however **NX_NONCOALSCEDFLAG** events are never coalesced regardless of the state of the **DPSSetTracking** flag.