Control Surfaces #1 - What they are and how they can improve your life

by David W. LeBlanc

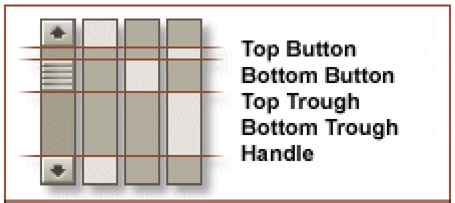
Welcome to the first installment in a series of articles devoted to the topic of control surfaces and how they can make your life better.

To begin, you might be wondering exactly what a control surface is. You're probably not alone.

Control surfaces began their life on the drawing board almost two years ago. They first manifested themselves publicly in the Photon 2.00 release. Originally the philosophy behind them was to provide developers with the ability to override the behavior of areas within a widget. However now they can be used for so much more.

Simply stated, control surfaces can be thought of as lightweight "widgets within widgets". Indeed, they deliver on the lightweight claim, weighing in at a mere 24 bytes apiece, while the meager **PtBasic** widget weighs in at a significantly heftier 300 bytes per instance, not counting any extra storage required by allocated resources.

Control surfaces are defined geometrical regions within a widget that can position, size and draw themselves. Additionally, they can define their own behavior. They do all this via callbacks and event handling flags that are supplied when the surface is created.



Sample Surfaces

It's important to note that control surfaces are a property of a widget; they require a widget in order to exist. However a widget may possess any number of control surfaces (currently the limit is 65535), making it possible to implement a whole UI using only one widget (say a **PtWindow**) at a fraction of the runtime data size (8% being a reasonable upper bound) as opposed to implementing the same UI using widgets.

This sounds pretty impressive. But before you begin re-implementing all your embedded photon apps using only control surfaces, you should be aware that there are, of course, some downsides. The widget library provides services to widgets which cannot, for reasons of economy, be provided to control surfaces. For instance, widgets have the concept of opacity which the library takes advantage of to reduce flicker at draw time. Control surfaces are simply drawn from the back to the front, without any regard to opacity. True, it would be possible to implement opacity in control surfaces, but that would require extra code and that responsibility lies with the application developer. Other widget level services not predicated on control surfaces include containment and focus.

Another major downside is that control surfaces are very raw elements and can only provide the behavior implemented by the developer creating them. While creating a "mock button" using a control surface is a fairly trivial matter, it would be prohibitive to implement **PtMultiText**'s functionality using a control surface.

Control surfaces are a fine tool, but like any other development tool, they have their place. And this is a topic that we will discuss in more detail in future installments. Stay tuned.

To get started, try compiling and running the following little demo application. Feel free to tinker around with it and browse the control surface documentation in the link to the right. The overview of the APIs should also give you a bit more exposure to the technical details of control surfaces.

Above all, have fun!

Overview of APIs
Sample.c (formatted)
Sample.c (unformatted)