MiscValueFieldPalette

The Basic Idea

This is a palette that provides two subclasses of a TextField.

The first, **MiscValueField**, adds two arrow buttons next to the field which allow the user to cycle through values with the mouse. The step values (regular and Alt click) can be set.

The second adds a SliderCell that coordinates with the TextFieldCell inside the **MiscSliderField**. The relative positions of the sub-cells can be set.

In both subclasses I also provide a way to allow for an expandable range. This *boundary* range is in effect when using the buttons or the slider but can be exceeded by typing values. This is useful, for example, if you have a line width value. You could make the boundary range from 1 to 20 points initially and still allow the user to type in 30 and get it. The buttons would then range from 1 to 30. You can still set absolute limits that cannot be exceeded. Using the same example, you could set the

lower limit of the line width to 0 ± the user could type in a fractional width or even zero, but couldn't get a negative value.

Both of these have been implemented entirely within TextFieldCell subclasses ± **MiscValueCell** and **MiscSliderCell**, respectively. This means they work equally well in a Matrix as they do by themselves. Each Cell can be configured independently.

Another major feature of these classes is that they will only send a single action when the value is adjusted. So you can scroll through many choices and only have to deal with the actual changed value once. It will also send an action every time the value changes if you want with a simple **setContinuous:** call.

The Extended Idea

For both subclasses there is also a way to connect an object that can act as a vendor of strings. By implementing a couple of simple methods and connecting the object, the **MiscXXXXCell**s will display an arbitrary string instead of a number. The methods were selected so a simple Interface Builder connection to a StringList object (as found in the MiniExamples) will work and make it a very simple matter to get a list of words to select from. This also allows simple sharing of lists within a .nib file. When an object is connected as a string vendor, the various limit values found in the IB inspector are ignored and the buttons will range throughout the entire list of provided strings. This is set up so you can simply fill a StringList with strings and know that all of them will be displayed without having to count them and keep all the various parameters in sync. Objects other than StringList will also work, they only need to implement two methods ± **MiscStringArray** implements the necessary methods but is currently not on a palette, so it's a little more trouble.

The Reason

I was writing a synthesizer patch editor and had whole bunch of places where I needed to have a particular range of values which might or might not be represented by names. I wanted a simple interface that would allow me to adjust the values without having to tab around and type them all in. I was in no real mood to actually connect up all those buttons and fields. I also didn't want to have to deal with getting an action message *every* time the value changed once when I was trying to get to a very different value. Thus the **MiscValueField**.

The **MiscSliderField** came about as a natural extension in design. I also wanted it for another project I have but I was going to put it off for a while (so I could work on actual programs instead of interface items). I was actually prodded into getting it done by Don.

The Inspectors

I added a bunch of methods in the **MiscXXXXField** that pass the requests through to the **MiscXXXXCell** for all the state information. This allows the same Attribute Inspector panel to be used when editing either object. Keeps things consistent and non-redundant.

A Neat Thing

I figured out a way to make my button icons survive with the **MiscValueCell** object through an archiving/unarchiving round by using named images. I set up the NXImages of the two arrows

using a name and setting them to retain their data. This forces them to write the image into the stream during an archive and because they are named they only go in once. I find this to be a great solution since the images are so small and it seems to be so easy to forget about them when working on a project. What this means is that the .tiff files can stay in the .palette and you only need to link against the library code. No subproject necessary, although it's there if you want it.