PSActionPallete implements an object class and InterfaceBuilder palette that can be viewed as a way to add simple scripting to InterfaceBuilder (using PostScript as the scripting language) or as a simple PostScript previewer object class.

The PSAction object responds to the standard *take*ValueFrom:* messages (e.g. *takeIntValueFrom:*) and puts the value received on top of PostScript's stack along with the PostScript code associated with the PSAction object (usually scripted from within InterfaceBuilder). This is executed and the PSAction object checks the datatype of the result left on PostScript's stack, pops it and sends the appropriate *take*ValueFrom:* message onto the PSAction object's own target. (Error messages generated in the PostScript execution can optionally be directed to a Text object.)

This gives you the ability to add PostScript code between a Control object and it's target, in InterfaceBuilder, to adjust the result to something more appropriate for your program to take as input. (You can adjust the output of a slider to be non-linear, round results to increments of 0.5, substitute strings like *high*, *medium* & *low* for numeric values, etc.)

Additionally, you can optionally associate a view with the PSAction object where PostScript graphics generated by the script will be rendered. This, for example, gives you a way to implement visual feedback to the user directly in InterfaceBuilder. (The **PSActionExamples** `*.nib` file mentioned below has simple examples of giving feedback about the radius of a circle and gray shade as determined by a slider.)

To use the PSAction object in your application, along with dragging the palette into InterfaceBuilder, you need to either include the sources for PSAction.{h,m} as well as PSActionWrap.psw in ProjectBuilder or include the PSAction.o and PSActionWrap.o object files. In general, since everything that the PSAction object needs can be set from

InterfaceBuilder, you don't need to reference the PSAction* header files from your own code unless you manipulate PSAction objects programmatically.   You'll also need to include dpsops (libdpsopts.a) in your application's list of libraries.

A quick example in 10 easy steps:

1) Start up InterfaceBuilder and select the '*Document>New Application*' menu item.
2) Drag the PSActionPalette into InterfaceBuilder's **Palettes** window.
3) Select the PSAction palette item and drag an instance into the *Objects* suitcase.
4) To 'My Window' add a Slider and two TextFields.
5) Connect the Slider to the PSAction instance, use the *takeFloatValueFrom:* method.
6) Connect the PSAction instance to one of the two TextFields (we'll call it A), use the PSAction object's *target* outlet.
7) Connect the PSAction to the other TextField (let's call it B), use the *script* outlet.   **Be sure to empty out the contents of this TextField!**
8) Make sure everything works correctly by running '*Document>Test Interface*'.   You should be able to move the slider and see TextField *A*'s value change between 0 & 1.
9) Set the content of the script, TextField *B*, to be '4 mul   cvi 4 div'.   You can even do this in test mode if you want.
10) Moving the slider in test mode should now produce values between 0 & 1 but in increments of 0.25.

Add another TextField connected to the PSAction *errors* outlet and experiment!

The PSAction class is a subclass of Object and has four outlets that can be set from InterfaceBuilder or via the following methods:

- script;
- setScript:anObject;

A Control or Text object or subclass.   This object contains the PostScript code that the PSAction object will execute.   If this outlet isn't set or the object is devoid of text, then the target/action message sent to the PSAction object will simply be relayed to the PSAction object's own target.   In normal usage, once the PSAction script is debugged, the text is set invisible and not selectable or keep visible in a window not accessible to the user of the program.   (You can of course leave the script editable if you're implementing an embedded PostScript previewer in your application.)   Though generally set from within InterfaceBuilder, the PostScript code contained in the object pointed to by this outlet can be changed by your program at runtime as well.

- target;
- setTarget:anObject;

A Control object or subclass (e.g. Slider, TextField).   This object receives a takeStringValueFrom:, takeIntValueFrom: or takeFloatValueFrom: message from the PSAction object depending on what it finds on top of the PostScript stack after the script executes.   (PostScript names are converted to strings and boolean types are converted to integers.)   The message sent to the target will not necessarily correspond in type to the message sent to the PSAction object itself.   (E.g. a slider can send the PSAction object a float but the PSAction object might pass an integer or a string to its own target based on what is found on the top of the PostScript stack.)   If this outlet is *nil* or doesn't respond to the standard take*ValueFrom: messages, no message is sent.

(Although there are target access methods, here are no *action* & *setAction:* access

methods in the PSAction class as the action performed depends on what is left on PostScript's stack after script execution.)

- errors;
- setErrors:anObject;

A Text object or subclass.   This optional outlet is where PostScript messages are sent if there errors executing the script.   (PSAction object internal error messages are sent here as well.)   This is primarily intended for debugging.

- view;
- setView:anObject;

A View object or subclass.   This optional outlet should be set to a custom view object in InterfaceBuilder if you want to see the graphics generated by the script, if any.   If not set, an invisible scratch view is used instead.

There are several flags that can be set from the InterfaceBuilder inspector for the PSAction object or via the following methods:

- (BOOL) isEnabled;
- setEnabled:(BOOL) flag;

Disables the PSAction object such that it won't execute the PostScript code nor pass a value onto its target.   You can make the PSAction object just pass on values handed to it by setting the script outlet to *nil*.   The object is enabled by default.

- (BOOL) isErrorReported;
- setErrorReported:(BOOL) flag;

Turns on and off error reporting in the PSAction object.   Turned on by default.   If there is no errors outlet set, error messages go to `stderr`.

- (BOOL) isCacheCleared;
- setCacheCleared:(BOOL) flag;

Determines if the view object pointed to by the view outlet is cleared before the script is executed.   The cache is cleared by default.   You probably don't need to worry about this unless you've set the optional view outlet in InterfaceBuilder.

The input to the PSAction object (the value that gets put on top of PostScript's stack and causes the script to execute) can be set from a target/action object within InterfaceBuilder or one of the following methods:

- takeIntValueFrom:sender;
- setIntValue:(int) anInt;

- takeFloatValueFrom:sender;
- setFloatValue:(float) aFloat;

- takeDoubleValueFrom:sender;
- setDoubleValue:(double) aDouble;

(The *DoubleValue*: methods really only handle the float type PostScript doesn't

have both a float and double type.)

- takeStringValueFrom:sender;
- setStringValue:(const char *) aString;

<u>The result of passing a value to the PSAction object and executing the script are queried with the following standard methods (usually by the *target* object):</u>

- (int) intValue;
- (float) floatValue;
- (double) doubleValue;   (See the comment about double precision numbers above.)
- (const char *) stringValue;

The **PSActionExample** directory contains an example InterfaceBuilder *.nib file that shows some potential uses of the PSAction object.   These can be examined and executed directly from InterfaceBuilder in '*Test Interface*' mode or you can build the application and run it.

<u>Miscellaneous:</u>

PSActionPalette includes a slightly modified version of the palette.make file so that it can correctly pass OTHERLINKED which is defined in the standard palette.make but never used.   This allows linking with libdpsops.a which is required and not already linked into InterfaceBuilder.

The current release of PSActionPalette was built and tested under the 3.2 release of NeXTSTEP -- use under earlier (or later) versions may require modification.