s an introduction to the Screen function, we provide some simple code with explanations to orient you to PTB's structure.

Here is an example of the functions mentioned in Intro to Psychtoolbox with supporting functions required to make a working program.

```
[w, wRect] = Screen('OpenWindow', 0, BlackIndex(0));
Screen('TextSize', w, 32);
Screen('TextFont', w, 'Helvetica');

DrawFormattedText(w, 'Hi, there', 'center', 'center', WhiteIndex(w));
Screen('Flip', w);
WaitSecs(2.0);

DrawFormattedText(w, 'How are you?', 'center', 'center', GrayIndex(w));
Screen('Flip', w);
WaitSecs(2.0);

picture = imread('smiley.jpg');
tex = Screen('MakeTexture', w, picture);
Screen('DrawTexture', w, tex);
Screen('Flip', w);
WaitSecs(2.0);

Screen('CloseAll');
```

First, we initialize the double-buffered window we will use for presentation. 0 is the screen number (monitor/display) we want to use for presentation. On a Windows OS, 0 refers to all displays. BlackIndex() is defining the background color for our new window as the standard black value for screen number 0 on the current system. The initialized window is saved as an output (*w* here, referred to as a window pointer). The size of the full window is saved as a 4 component vector, stored in wRect here. Let's say the vector looks like *{0 0 1680 1050}* . Each number represents the pixel *x* or *y* value of the window, specifically *{left (x1), top (y1), right (x2), bottom (y2)}*. In our example, *{0, 0}* would be the coordinates of the top left corner, and *{1680, 1050}* would be the coordinates of the bottom right corner. Most of the time, you won't need to use wRect at all, but we will give an example of where it can be used below.

Next are a couple of commands to set the font and font size of text for our new window *w*. Then, the DrawFormattedText() command tells PTB to write the text 'Hi, there' in the vertical and horizontal center of the back buffer (automatically the back buffer) of *w* in white text. Note, the BlackIndex(), WhiteIndex(), and GrayIndex() functions will accept either a screen number (here is 0) or window pointer (here is *w*).

As soon as the Screen('Flip') command is executed, the screen with 'Hi, there.' becomes visible to the viewer. Then, we prepare another message for presentation, 'How are you?' in gray text. To keep this screen from replacing the first message after only a handful of milliseconds (the next monitor refresh), we threw in a WaitSecs() command to ensure 'Hi, there' is presented for 2 seconds before Screen('Flip') replaces it with 'How are you?'. Similarly, we show this message for 2 seconds before moving on to the next set of changes to the screen.

After the text commands, we have commands that work in conjunction to display a picture on the screen. First, we load the image into a matrix that represents colors by pixel. This process is accomplished by imread(). Next, we prepare the image to be drawn onto the back buffer; think of the Screen('MakeTexture') command as creating a stencil so that drawing the image on the dry erase board will take much less time. Lastly, the picture is drawn onto the back buffer using the Screen('DrawTexture') function. The screen is flipped and the image is displayed for 2 seconds.

The Screen('DrawTexture') function can be run with many extra inputs to change the display settings of the picture. After *w* and the texture input, the fourth input to the function can be a vector specifying how much of the image to display. For example, if you only want to show the top left 50x50 pixel square of the image, you can put in *{0 0 50 50}* as the fourth input. Similarly, the fifth input specifies where on the screen you want to display the image. This is where wRect comes in useful. Say you only want to show the picture on the top half of the screen. You can input *{0 0 wRect(3) wRect(4)/2}* , where the second *y* value is divided by 2, implying that this new rectangle only goes halfway down the screen.

The last line in our example above is required to properly close the window and anything drawn on it. This call also quits PTB. Generally, you want to call Screen('CloseAll') at the end of your script, but there is also a function called Screen('Close'). This call is useful if you have an excessive number of Screen commands in succession (like in a for loop for an event-based paradigm). Like marker residue on a dry erase board, it seems as though PTB doesn't do a perfect job of erasing the back buffer before drawing the next item onto it. Thus, multiple Screen calls can create a buildup of memory, causing commands to slow down. Adding Screen('Close') at the end of the for loop will properly close all previously drawn objects while keeping the front buffer intact.