Tablet Demo README

Author and responsible contact: Andrew D. Wilson (adwkiwi@gmail.com)
(this text edited by Mario Kleiner with a few more comments and additions)

The code and examples in the PsychContributed/Wintab/ subfolder of Psychtoolbox is provided mostly by Andrew D. Wilson. It is not officially maintained or supported by the Psychtoolbox developers, but provided "as is" for now.

Please also follow the discussion on the Psychtoolbox forum for more infos:
http://tech.groups.yahoo.com/group/psychtoolbox/message/9296

This README walks you through the demo code and various features of the WinTabMex Matlab MEX file addition to the Psychtoolbox-3. WinTabMex allows you to interface directly with a graphics tablet on Microsoft Windows from within 32 bit versions of Matlab. The vendor provided tablet driver needs to support the Windows WinTAB – API, but this can be taken for granted for all common graphics tablets. You may need to install proper drivers from your tablet manufacturer though. This MEX file has been successfully tested with some Wacom tablets so far.

WinTabMex.mexw32 is the file you will need if you have Matlab Version 7.4 (R2007a) or later. The .mexw32 is a 32-bit Matlab file that, similar to Screen, allows you to use Matlab to talk to the tablet driver. For use with older Matlab versions (5.0 up to 7.3), you will need to put WinTabMex.dll into your Matlab path instead.

The following link points you to the download location of Wacoms tablet drivers:
Wacom Tablet Driver (click to follow the link)
If the drivers are not properly installed, Matlab will fail to load the MEX file with some kind of "Invalid MEX file" error or some similar error message.

This folder also contains the C language source code to the Wintabmex driver. If you need to customize or improve the driver, follow the instructions at the top of the WinTabMex.c source file. If you make significant improvements, please consider donating them back to the Psychtoolbox project for the benefit of others.

WinTabMex has 6 functions with the following syntax: (Similar usage information will be displayed if you just type WinTabMex at the Matlab command prompt)

WinTabMex(0, wPtr) You must make this call to initialise the driver and give you access to the rest of the functions. This is conceptually like a call to Screen('OpenWindow'), in that this is the first thing that should happen.

Note you need a window pointer wPtr, and so you must have already made a call to wPtr = Screen('OpenWindow') beforehand to acquire such a wPtr from Psychtoolbox. Alternatively you can pass in the name displayed in the title bar of any other open window on your desktop. The window can be of any old size. If you don't need the window to display anything, you can call WinTabMex(0, wPtr, 0) to hide the unneeded window .

WinTabmex(1); This is the last thing you must call, and it shuts down Matlab's access to the tablet driver. This is a clean up feature analogous to Screen('CloseAll').

WinTabMex(2); This clears the tablet event queue from stale tablet events (see later) and readies the tablet for data collection, ie. It restarts data collection again. Should be called at the beginning of the response period of a trial.

WinTabMex(3); Stops/pauses data collection. This should be called at the end prior to WinTabMex(1), or at the end of a trial to pause data collection until start of the response period of the following trial

WinTabMex(4) = 50; Changes the event queue size. Default is 50 events (500ms of data at 100Hz) and although you can increase this, there are limits and tradeoffs which will vary from system to system. If you change the queue size you must always test the returned old- and new size of the queue: [old, new] = WinTabMex(4, wantedSize);

pkt = WinTabMex(5); Pops the oldest data packet 'pkt' from the event queue (first-in-first-out). Repeated calls to this function will retrieve the actual tablet event packets.

pkt is an 9x1 column vector consisting of

pkt(1)	x axis position (tablet coordinates)
pkt(2)	y axis position (tablet coordinates)
pkt(3)	z axis position (tablet coordinates)
pkt(4)	encoded button state (works a little erratically)
pkt(5)	an index of which data point this is since the last call to WinTabMex(2)
pkt(6)	timestamp in ms from the tablet (the difference between these values is the sample rate; 10 would be 10ms or 100Hz)
pkt(7)	signals various events (eg penOutOfRange)
pkt(8)	flags what has changed since the last sample
pkt(9)	Normal pressure: Info about the pressure along the tablet surface normal, or pressure on the pen tip, in arbitrary units. Value range and meaning of this parameter depends on the tablet device. Some tablets may not report any meaningful values here.

The basic chain of events in using the driver is:

```
%%%%%Pseudo Matlab Code%%%%%

wPtr = Screen('OpenWindow', 0, 0, [0 0 30 30]);


WinTabMex(0, w);     %Initialize driver, connect it to window ‘wPtr’.

WinTabMex(2);        % Start data acquisition.


for time = start:stop

        pkt = WinTabMex(5);

        if ~isempty(pkt)

                disp(pkt);

        end;

end


WinTabMex(3);        % Stop/Pause data acquisition.

WinTabmex(1);        % Shutdown driver.


Screen(‘CloseAll’);

%%%%%/Pseudo Matlab Code%%%%%
```

There are two ways to get data from the event queue:

Fast loops: As soon as an event is detected after a call to WinTabMex(2), where an event is some change in the stylus status, the event queue begins to fill up. I use a Wacom intuos[2] tablet with a sample rate of 100Hz, and the event queue is (default) 50 data points long (alterable with WinTabMex(4) but with varying levels of success). The queue can therefore contain pkts from a 500ms period. After this time, the queue will begin to overflow, until calls to WinTabMex(2) empty it out below it's limit.

I use this to record data during stimulus presentation. I have an experiment with 280ms of stimuli to present, and immediately before this begins I clear the queue with a call to WinTabMex(2); the first data point in the queue is then timelocked to first stimulus. Once I'm finished presenting stimuli, I run a **fast loop** to drain the queue as fast as possible; see fastLoop.m for a demo.

fastLoop.m – place the sylus on the tablet when told, and leave it there or move it around after pressing the space bar. A screen will tell you the queue is filling; this screen is up for 3s just so you aren't rushed, but you will only end up with data from the last 500ms of this time period. The queue will then drain – the key is that the data drained is from the first period, not real time.

Slow loops: I then want to record data as people move in response to the stimuli. If nothing else is happening (i.e. nothing being drawn, etc) I can simply run a loop for a set amount of time, in which I call WinTabMex(5) at least once per sample time interval (10ms at 100Hz) while the movement is occurring. See slowLoop.m for a demo, in which I actually call WinTabMex(5) as fast as I can until a) I get a data point or b) my 10ms is up; if the latter I add a row of 0s to the data to index a missed data point). Because you can poll the tablet faster than it can create a pkt, you need to do this sort of thing to ensure you don't accidentally ask for the data too soon. There are other solutions but this is the basic problem.

slowLoop.m – place the sylus on the tablet when told, and leave it there or move it around after pressing the space bar. A screen will tell you the tablet is recording what you are currently doing; this data will then get output to a .mat and .xls file. There will often be no data right at the start – you need to pay attention to the fact that the event queue won’t start to fill until some change is detected.

So fast loops are for emptying out a queue that formed while you were busy doing things like calls to Screen(‘Flip’), etc. Slow loops are for real time data acquisition when you don’t have anything else to do.