

Chapter 9 – Key Presses

Here we will go through the various ways that you can collect keypresses and discuss their strengths and weaknesses. Different techniques are useful for different purposes. This table should help you keep these commands straight in your head.

	Psychtoolbox windows	Timing	What was pressed	Waits for keypress	NOTES
<code>pause</code>	NO	NO	NO	YES	Just waits for a specified period of time or for any key to be pressed on the keyboard
<code>input</code>	NO	NO	YES (character)	YES	Just waits for a key to be pressed on the keyboard and returns the character or number of that key
<code>CharAvail</code>	YES	NO	NO	NO	Simply checks whether there is a key press in the event queue
<code>GetChar</code>	YES	Dubious accuracy	YES (character)	YES	Checks or waits for a key press in the event queue. Returns what the key was, and when it was pressed
<code>KbCheck</code>	YES	Good	YES (key)	NO	Tests whether a key has been pressed at that moment in time
<code>KbWait</code>	YES	Good	NO	YES	Waits for a key press to occur

Collecting keypresses using `pause` and `input`

These techniques are useful when you are not using Psychtoolbox. Remember that none of these techniques have good timing. It's easiest practicing these commands in a m-file, since many of these commands will accept Return as a key press (if you type them into the command line, the Return you use at the end of the line will also be used as the input into the key press command).

>pause

This is the simplest command you can use to collect a key-press. It simply pauses the computer until a key (any key) is pressed on the computer and doesn't collect what that response is.

Bear in mind that the command window needs to be in front for the keypress to be available to Matlab, so `pause` doesn't work well if you are using Psychtoolbox. In that case use `GetChar` instead.

Pause can also be used to enforce a delay - `pause (3)` pauses for 3 seconds.

>input

This command again pauses the computer until a key is pressed but it allows you to collect the response. By default the response is a number

```
>resp_num=input('press a number key ...')
```

But you can also specify that `input` will accept a string, as shown below. In that case if the subject inputs a number key the computer will assume that the subject chose the character '3' rather than the double 3.

```
>resp_char=input('press a number key ...', 's')
```

```
>whos
```

Note that `resp_num` is a double and `resp_char` is a character.

If in a program you want to only accept a certain type of response, then you need to write a loop like this one:

```
1      % PracticeKeyPresses
2      % a program to practice different ways of collecting
3      % key press responses
4      %
5      % written if 4/2007
6
7      % using input
8      disp('Using input command')
9      resp='x';
10     while resp~='a' & resp~='b'
11         resp=input('press a or b ... ', 's');
12     end
13     resp
14
```

Bear in mind that the command window needs to be in front for the key-press to be available to Matlab, so `input` doesn't work well if you are using Psychtoolbox. In that case you will need to use `GetChar`, `CharAvail` or `KbCheck` instead.

Collecting keypresses using GetChar & CharAvail

When you type into the keyboard (or any input device) the key presses get saved into an event queue. This is why sometimes when you type really fast and the computer is also busy with other tasks there will be a pause before the letters suddenly appear in your document. `CharAvail` and `GetChar` commands read from the event queue. It's therefore important to empty events from the event queue before collecting responses using these two commands

>FlushEvents

Before using any of the following commands.

>CharAvail

`CharAvail` looks to see if there is anything in the event queue.

```
[avail, numChars] = CharAvail;
```

`avail` will be 1 if characters are available, 0 otherwise. `numChars` may hold the current number of characters in the event queue, but in some system configurations it is just empty, so do not rely on

numChars providing meaningful results, unless you've tested it on your specific setup. If `avail` is 1 then you need to call `GetChar` to find out what the actual key press is – `CharAvail` just tells you whether there was a key press.

NOTE: When BACKGROUNDING is enabled, Matlab removes all characters from the event queue before executing each Matlab statement, so `CharAvail` always reports 0. So for this to work you need to make sure that BACKGROUNDING IS OFF (using the preference setting in the menu bar is probably easiest).

```
15 % CharAvail
16 WaitSecs(1)
17 Screen('Preference', 'Backgrounding', 0);
18 disp('Using CharAvail command')
19 disp('Wait 2 sec to see if you pressed a key during that
20 time');
21 FlushEvents
22 tic
23 while toc < 2
24 ;
25 end
26 resp = CharAvail;
27 resp
```

`CharAvail` is useful when you want to see whether your subject pressed a key while you were doing something else (such as displaying images) in the meantime. It's especially useful if you are doing something like fMRI where you don't want to wait indefinitely for a subject response.

>GetChar

If there is something already in the event queue then `GetChar` retrieves it. If there isn't something in the event queue then `GetChar` waits for a typed character and lets you save the response.

It's used as follows:

```
[resp, when] = GetChar([getExtendedData], [getRawCode]);
```

`char` is the character that was typed

`when` is a structure. It returns the time of the key press, the "adb" address of the input device, and the state of all the modifier keys (shift, control, command, option, alphaLock) and the mouse button. If you have multiple keyboards connected, `address` may allow you to distinguish which is responsible for the key press. `when.secs` is an estimate of the time when the key press happened. However the timing of `GetChar` is not necessarily reliable, the reported values can be off by multiple dozen or even hundreds of milliseconds. If you are interested in precise timing you should check the timing of `GetChar` on the particular system that you are using or use `KbWait` or `KbCheck` instead.

`getExtendedData` tells `GetChar` whether or not to collect when data, if set to 0 then `GetChar` will be a tiny bit faster. `getRawCode` determines whether you want to return the character information in ascii or char (the default) format. Normally you won't bother using either of these input arguments.

Add the following lines to your `PracticeKeyPresses` m-file. If you press a key `GetChar` behaves very like `CharAvail` except that it retrieves what the character is. If you don't then `GetChar` waits

```
27 % GetChar
28 WaitSecs(1)
```

```
29     disp('Using GetChar command just on its own')
30     FlushEvents
31     tic
32     while toc<1
33         ;
34     end
35     [resp_GC, when_GC]=GetChar;
36
37     disp('now you have pressed a key')
38     resp_GC
39
```

Note that even though the command line may appear, the program doesn't actually continue to print 'now you have pressed a key' until you press a key – if there's nothing in the event queue then GetChar will wait for a response.

What if you only want to accept certain responses (e.g. the 'm' or 'f' keys)?

```
40     % GetChar - only accepting certain responses
41     WaitSecs(1)
42     disp('Using GetChar command - only accepting certain
43     responses')
44     FlushEvents
45     resp_GC2='x';
46     while resp_GC2~='m' & resp_GC2~='f'
47         disp('press m or f ... ');
48         resp_GC2=GetChar;
49     end
50     resp_GC2
```

What if you want to combine the useful property of CharAvail that it doesn't wait indefinitely for a response with the useful property of GetChar that you can find out what the response was?

```
51     % Combining GetChar and Char Avail
52     WaitSecs(1)
53     disp('Combining GetChar and Char Avail')
54     disp('Wait 2 sec to see if you pressed a key during that
55     time');
56     FlushEvents
57     tic
58     while toc<2
59         ;
60     end
61     resp_CA3= CharAvail;
62     if resp_CA3==1
63         resp_GC3=GetChar;
64         disp(['You pressed key', resp_GC3])
65     else
66         disp('You did not press a key');
67     end
```

One more thing to remember

KbCheck and KbWait are MEX files, which take time to load when they're first called. They'll then stay loaded until you flush them (e.g. by changing directory or calling CLEAR MEX). So your timing on the first trial will be more accurate if you just call them at the beginning of the program.

Collecting keypresses using KbCheck & KbWait

`KbCheck` and `KbWait` don't pull key presses out of the event queue. Instead they monitor the state of the keyboard at that very moment. The advantage of this is that they tend to have more accurate timing on some systems, and there is no lag – the minute the key press happens the `KbCheck` and `KbWait` command knows that the key press has happened. The disadvantage is that it's hard to do something else (e.g. display images) while constantly monitoring the state of the keyboard.

One thing that is a little weird about `KbCheck` and `KbWait` is that they actually refer to the key that was pressed on the keyboard rather than the character represented by that key. This is because they are lower level commands. What character a key represents actually varies across computer systems, which means you need to convert your key press information into character information.

While most keynames are shared between Windows and Macintosh, not all are. Some key names are used only on Windows, and other key names are used only on Macintosh. For a lists of key names common to both platforms and unique to each see the comments in the body of `KbName.m`.

`KbName` will try to use a mostly shared name mapping if you add the command `KbName('UnifyKeyNames');` at the top of your experiment script. In fact, **KbName often won't work unless you add this line to the beginning of your scripts!**

Psychtoolbox has a great demo for `KbCheck` and `KbWait` which is called `KbDemo`. You can try it by just typing `KbDemo` at the command window:

```
>KbDemo
```

This is also a good way of making sure that `KbCheck` is working. If not, try replacing the version of `KbCheck.m` that you have with a revised version found here:

<http://en.wikibooks.org/wiki/Psychtoolbox:KbCheck>

The examples below are simplified versions of taken from `KbDemo`

KbCheck

Checks to see whether a key on the keyboard is pressed down at that very moment in time.

```
[keyIsDown, secs, keyCode] = KbCheck;
```

`keyIsDown` is 1 if *any* key, including modifiers such as <shift>, <control> or <caps lock> is down. 0 otherwise.

`secs` is the time of the key press as returned by `GetSecs`. This is an accurate way of getting timing but you should still read the help file for `KbCheck` and `GetSecs` carefully if your experiment depends on accurate timing.

`keyCode` On Macs this is a 128-element array, on PCs it is a 256 . Each number in the array represents a keyboard key. If a key is pressed that index in the array is set to 1, otherwise it will be set to 0. To convert a `keyCode` to a vector of key numbers that were pressed use `find(keyCode)` . To find out what actual key that was use `KbName`.

KbWait

Just like `KbCheck`, but it waits until a key on the keyboard is pressed down and simply returns the time that the key was pressed.

```
secs = KbWait;
```

`secs` is the time of the key press as returned by `GetSecs`.

KbName

KbName maps between KbCheck-style keyscan codes and the character that key represents. Use KbName to make your scripts more readable and portable, since keycodes, are cryptic and vary between Mac and Windows computers.

```
kbNameResult = KbName(arg)
```

KbName actually will let you go either way from keycodes to characters, or vice versa. If you send in as input a string designating a character then KbName returns the keycode for that character.

```
>KbName('t')
```

If on the other hand you send in a number representing the keycode, then KbName will return the character of that keycode

```
>KbName(84)
```

The numbering of these practice examples matches their order in KbDemo, but I'm going through them in order of how complicated they are. That's why the naming has a funny order (we start with KbPractice3)

```
1      % KbPractice3
2      % Wait for a key with KbWait.
3      % a simplified version of KbDemo by IF 4/2007
4
5      WaitSecs(0.5);
6      disp('Testing KbWait: hit any key. Just once.');
```

Line 8-10. KbWait returns the time in seconds (with high precision) since the computer started. Usually we don't want to know the absolute time in seconds, but the time since some other event (e.g. since you put an image on the screen). So here we are calculating the time between the key press and an arbitrary start time.

```
1      %KbPractice1
2      % Displays the key number when the user presses a key.
3      % a simplified version of KbDemo by IF 4/2007
4
5      WaitSecs(0.5);
6      disp('1 of 4. Testing KbCheck and KbName: press a key to see
7      its number');
8      disp('Press the escape key to exit.');
```

Chapter 9 - 7 - 4/22/2007

```
19     % then display its code number and name.
20     if keyIsDown
21
22     % Note that we use find(keyCode) because keyCode is an array.
23     str=['You pressed key ', find(keyCode), ...
24     ' which is ', KbName(keyCode)];
25     disp(str)
26
27     if keyCode(escapeKey)
28         break;
29     end
30
31
32     % If the user holds down a key,
33     % KbCheck will report multiple events.
34     % To condense multiple 'keyDown' events into a single event,
35     % once a key has been pressed
36     % we wait until all keys have been released
37     % before going through the loop again
38     while KbCheck; end
39     end
40 end
```

Line 11. This is a loop that flips around doing anything as long as KbCheck reports back that a key is pressed on the keyboard. Means that the program doesn't continue until all keys have been released.

Line 13 while 1 is always true, so this loop will continue indefinitely. **Line 28** forcefully breaks us out of the loop if the escape key is pressed.

Line 14 See if a key is currently pressed on the keyboard. If not, we skip the next for loop from **lines 20-38**, and basically check again almost immediately.

Line 23 Display which key has been pressed.

Lines 27-29 If the key that has been pressed is the escape key break out of all loops including the indefinite while loop.

Line 37. Once again make sure that all keys have been released before checking to see if a key has been pressed.

```
1     %KbPractice2
2     % Displays the number of seconds that have elapsed
3     % when the user presses a key.
4     % a simplified version of KbDemo by IF 4/2007
5
6     WaitSecs(0.5);
7     disp('Testing KbCheck timing: please type a few keys. (Try
8     shift keys too.)');
9     disp('Type the escape key to exit.');
```

```
10
11     escapeKey = KbName('ESCAPE');
```

```
12
13     startSecs = GetSecs;
```

```
14
15
16     while 1
17         [ keyIsDown, timeSecs, keyCode ] = KbCheck;
18         if keyIsDown
19             str=[ KbName(keyCode), ' typed at time ', ...
20             num2str(timeSecs - startSecs), 'seconds'];
21             disp(str)
```

```
22
23         if keyCode(escapeKey)
24             break;
25         end
26
27     % If the user holds down a key,
28     % KbCheck will report multiple events.
29     % To condense multiple 'keyDown' events into a
30     % single event, we wait until all
31     % keys have been released.
32         while KbCheck; end
33     end
34 end
```

Line 19-22 This example is very similar to that of KbPractice1 except here we report both what key was pressed, but also when it was pressed.

Summary

There are two ways to collect responses.

One (CharAvail/KbCheck) is to collect them from the event queue where they are stored. The advantage of that is you can do something interesting (display images) and then just pull the responses that the subject was making while you did that interesting thing from the event queue whenever you have time. The disadvantage is that these commands may have limited temporal accuracy.

The other way (KbCheck/KbWait) is to collect them directly from the keyboard. This has better temporal accuracy. But if you aren't monitoring the keyboard at that moment in time (e.g. because the program is busy displaying images) then the computer will not notice the key press. Because key presses are actually very slow compared to most programming commands you can actually put some interesting stuff inside the loop that is monitoring the keyboard without missing key presses. You can see how this works by adding the following commands between line 17-18 in Practice 2, e.g.

```
img=rand(100);
img=img.*rand(100);
```

But you can't put too much in the loop without beginning to miss key presses. You can check this by making the stuff inside the loop more time consuming

```
img=rand(10000);
for i=1:10
img=img.*rand(10000);
end
```

How much stuff you will get away with will depend on the speed of your computer.