Mex-File Plug-in for Fast MATLAB Port I/O Access (32-bit MATLAB on 64-bit Windows XP, Vista, 7)

As of April 2010, the student version of MATLAB installs as a 32-bit application...even on 64-bit Windows. This document describes how to setup and run the 64-bit inpoutx64.sys driver on 64-bit Windows and then access it via the IO32 mex file interface in a 32-bit MATLAB application.
A version of this software for 64-bit MATLAB on 64-bit Windows can be found here.
A version of this software for 32-bit Windows operating systems can be found here.

Windows Vista and Windows 7 (32-bit users) should note the **Vista Installation Notes** near the end of this document.

In order to accomplish very fast port I/O using a NO COST add-on to MATLAB, we have developed a C++ extension (mex-file) that uses native methods to access low-level hardware. This 32-bit mex-file is named **io32.mexw32**. It uses a freeware self-installing kernel-level system driver that is exported from a dynamic link library named **inpout32a.dll**. [Note: Self-installation of the driver requires that the MATLAB be run with Administrator privileges. The driver must have been previously installed in order to support non-Administrator users].

Once the required software modules are installed in the appropriate file directories, you can use **io32()** to read and write to I/O port locations anywhere in the 64K address space. A simple benchmark test (iotimer_io32b.m or iotimer_io32.m) reveals that port I/O latencies of approximately 0.010 msec (i.e., 10 microseconds) can be achieved from within MATLAB using this approach.

To install this expanded capability: download the **io32.mexw32** module and move it to a directory in your MATLAB path (e.g., c:\cog2000\Cogent2000v1.29\Toolbox in the case of the USD PSYC 770 standard Cogent 2000 installation specification). Users of older versions of MATLAB 7 should rename **io32.mexw32** to **io32.dll** (since 32-bit mex files used to use the .dll suffix rather than the more informative .mexw32 suffix currently in use). Next, download the **inpout32a.dll** module and move it to the C:\windows\sysWOW64 directory.

io32() Command Usage Summary:

object = **io32**;	Calling **io32** with no input arguments from the MATLAB command window creates an instance of the **io32** interface object and returns a pointer to its location. This command must be issued first since the object pointer is a required input argument for all other calls to **io32**. This **io32** call will not work properly unless a return variable is specified (i.e., 'object' in the example to the left).
	Calling **io32()** using one input argument and a single return variable causes the *inpoutx64.sys* kernel-level I/O driver to be extracted

status = **io32(** object **)**;	from *inpout32a.dll* and automatically installed (i.e., no manual driver installation is required). object is the pointer to a previously created instance of io32 (see the step performed above); and, status is a variable returned from the function that describes whether the driver installation process was successful (0 = successful). Subsequent attempts to perform port I/O using io32() will fail if a non-zero status value is returned here. This step must be performed prior to any subsequent attempts to read or write I/O port data.
io32(object, address, data **)**;	Calling **io32()** with three input parameters allows the user to output data to the specified I/O port address. object is the pointer to a previously created **io32** object (described above); address specifies the physical address of the destination I/O port (<64K); and, data represents the value (between 0-255) being output to the I/O port.
data = **io32(** object, address **)**;	Calling **io32()** using two input arguments and one return variable allows the user to read the contents of the specified I/O port. object is the pointer to a previously created instance of **io32** (see above), address specifies the location of the I/O port being read; and, data contains the integer-format value returned after reading the I/O port.

The following MATLAB command snippet demonstrates how to use the **io32()** extension:

```
%create an instance of the io32 object
ioObj = io32;
%
%initialize the inpoutx64 system driver
status = io32(ioObj);
%
%if status = 0, you are now ready to write and read to a hardware port
%let's try sending the value=1 to the parallel printer's output port (LPT1)
address = hex2dec('378');          %standard LPT1 output port address
data_out=1;                        %sample data value
io32(ioObj,address,data_out);   %output command
%
%now, let's read that value back into MATLAB
data_in=io32(ioObj,address);
%
%when finished with the io32 object it can be discarded via
%'clear all', 'clear mex', 'clear io32' or 'clear functions' command.
```

MATLAB Scripts to Simplify Port I/O

The code examples above reveal that using the **io32()** extensions is a bit complex. In an attempt to reduce this complexity, a set of MATLAB scripts has been developed to simplify I/O programming.

In order to have access to these scripts: download the io32.mexw32, config_io.m, inp.m and outp.m files and move them to a directory in your MATLAB path. In addition, download the **inpout32a.dll** module and move it to the C:\windows\sysWOW64 directory as previously described above.

MATLAB I/O Script Usage:

config_io;	Installs the *inpoutx64.sys* driver required to access low-level hardware. This command must be given prior to any attempts to use the custom **inp()** or **outp()** scripts.
outp(address, byte);	This function writes the 8-bit value passed in the variable named byte to the I/O port specified by address.
byte = inp(address);	This function read the I/O port location specified by address and returns the result of that operation.

A simple benchmark (iotimer_inp.m) reveals that I/O using these scripts is significantly slower than calling the **io32()** object directly (as demonstrated above). Instead of being able to read a port with a latency of approximately 10 microseconds, using the **inp()** script yields a latency of approximately 40 microseconds. This is fast enough for many experimental psychology applications (such as scanning a button box, etc.). Use direct calls to **io32()** if your application requires the shortest possible I/O latencies (e.g., updating an analog output stream).

The following MATLAB code snippet demonstrates how to use the m-file I/O scripts:

```
%initialize the inpoutx64 low-level I/O driver
config_io;
%optional step: verify that the inpoutx64 driver was successfully installed
global cogent;
if( cogent.io.status ~= 0 )
   error('inp/outp installation failed');
end
%write a value to the default LPT1 printer output port (at 0x378)
address = hex2dec('378');
byte = 99;
outp(address,byte);
%read back the value written to the printer port above
datum=inp(address);
```

Windows Vista and 7 Installation Notes

Although our lab does not yet have much experience with Windows Vista, we were able to successfully install the software described above using the procedure described below (using MATLAB 7.5.0-R2007b and MATLAB 7.7-R2008b):

1. Log in as a user with Administrator privileges.
2. Disable UAC (User Account Control). An easy way to do this in Windows Vista is to: Start-Run-MSCONFIG. Select the Tools tab, scroll down to the option for "Disable UAC" and select it. Next, press the "Launch" button. You must then RESTART the system for this change to take effect.
3. Download and copy the inpout32a.dll file to the C:\windows\sysWOW64 directory.
4. Download the io32.mexw32, config_io.m, inp.m and outp.m files to a working directory of your choice. This directory will be added to your MATLAB path in step-6 below.
5. Start MATLAB in "Run as Administrator" mode (Right-click icon and select "Run as Administrator").
6. Add the directory containing the downloaded m-files to your MATLAB path via the File|Set Path|Add with Subfiles... menu command.
7. Run "config_io" from the MATLAB command window. If there's no error message at this point, you've successfully installed the software.
8. Optional: If you need to re-enable UAC (User Account Control), follow the instructions in step-2 but select "Enable UAC" instead of "Disable UAC".

Parsing Individual Bits within an I/O Byte

When one reads an I/O port one is usually interested in the status of a single bit among the 8-bits returned by a call to **inp(address)**. MATLAB provides a number of functions to deal with data on a 'bitwise' basis. For example, the following lines of code show how to test the status of a single input line using the **bitget()** function:

```
% Read current value of an input port at the specified address
% Note that the value returned by inp(address) is coerced into an 8-bit format using uint8
response = uint8( inp(address) );
% Take some action if the least-significant-bit is currently at logical-0 level
if (bitget( response,1) == 0)
   display('Input is active')
end
```

See also: bitset(), bitand(), bitor(), bitxor() for additional bitwise operators

---

Additional information about the *freeware* INPOUTX64 driver for 64-bit Windows XP/Vista/7 can be found here.
Special thanks to Phil Gibbons (www.highrez.co.uk) for providing the signed 64-bit version of the inpoutx64.sys kernel-level driver.
Versions of this software for 32-bit Windows systems can be found here

Last revised: 3 May 2010