1. General

1.1. What is the purpose of this FAQ?

This FAQ describes details and common problems with adding tablet support for Wacom tablets. It is primarily concerned with discussing issues related how to use the Wintab tablet application programming interface (API) to acquire data from Wacom’s currently shipping tablet series. Some of the information in this FAQ is applicable to all Wintab applications in general.

1.2. Who is responsible for this FAQ?

This FAQ is maintained and copyrighted by Wacom Technology. It may be freely copied and distributed. If you have questions concerning it, contact developer@wacom.com.

1.3. What tablet models does Wacom make and what do I have to know about differences between them to support them in my code?

Currently, Wacom ships a higher-end, professional tablet series called Intuos and a consumer tablet series named Bamboo. Wacom's web site at http://www.wacom.com describes in detail the models that Wacom currently sells.

There is very little you must do to support virtually all Wacom's tablet models, from those that are over five years old to the most recent. Once you have added Wintab support for one tablet, your program should work with virtually any Wacom tablets if the user has a recent version of our driver software. Except for a few features, such as two-handed input (DualTrack), your code should also work without change with other tablet manufacturer's tablets too, if they use Wintab.

The main issue you must account for to support all tablets is that some tablets will not support some data items (such as tilt or rotation) and the ranges of data items differ between tablet models. You must use the Wintab function WTInfo( WTI_DEVICES, wIndexOfData, pData ) to determine if a tablet supports a feature and the values which will be returned for that feature.

1.4. Why do Wacom tablets require using the Wintab tablet API to program the tablet?

Wintab is the industry standard for acquiring data from a graphics tablet on Intel machines and has been the standard for a number of years.

Unfortunately, there is no cross-platform graphics tablet API.


1.5. What resources are available help me write code to support Wacom tablets using Wintab?

Here are some resources which hopefully will help you (see the example set at: http://www.developer.wacom.com)

The PressureTest example in the Wintab SDK works well as a template to getting tablet support working in your application.

The TiltTest example program from shows how to get tilt data from Wacom tablets.

The CursorMaskEx example program from shows how to acquire data from two cursors for Wacom Intuos tablets. The source code has many comments about how to use Wintab.

Wacom will be more than happy to answer any questions concerning using Wintab with Wacom tablets. Contact developer@wacom.com.

1.6. With the older Wacom tablets, my program didn't use Wintab to acquire tablet data and instead read data directly from the tablet. Can I use data directly from the newer Intuos tablets or do I have to use Wintab?
In almost all cases, you should use Wintab. Unlike the older tablets, our newer Intuos tablets were designed so that the only way applications would communicate with the tablet would be through whatever tablet API is supported by the operating system under which the tablet is running. In the case of Windows, this API is the industry standard called Wintab. You can get a copy of the Wintab spec and the Wintab SDK from [http://www.developer.wacom.com](http://www.developer.wacom.com).

A very few individuals doing such things as university research have signed nondisclosure agreements (NDAs) with Wacom to get the data format and other information needed to communicate directly with the tablet. The data packets for Intuos are considerably more complex than the data format for the older tablets and may well change in the future. There is generally no good reason to directly communicate with the tablet instead of using Wintab, but if you have a unique situation, you can contact Wacom’s Product Management (developer@wacom.com) to see about signing an NDA.

1.7. How does the Wacom Wintab manage the context overlap order differently than other Wintabs?

Wintab manages an overlap order of tablet contexts. Only one context can receive a given tablet event. In general, if context A is above context B in the overlap order, context B will not get any tablet events. The Wacom Wintab manages the context overlap order differently than other Wintabs. The two differences are:

In other Wintabs, events fall-through to an underlying context if the overlying context does not handle the event. In the Wacom Wintab, events only fall-through to the next, underlying context if the underlying context is owned by the same application as the context above it. For example in other Wintabs, a context might not handle an event if the cursor is outside the context's tablet input area (re: lcInOrg and lcInExt, see wintab.h).

In other Wintabs, applications must manage their overlap order themselves. The problem is that many applications do not properly manage their overlap order, so two Wintab applications (even two instances of the same application) often do not work correctly in these Wintabs. The Wacom Wintab automatically rearranges the overlap order of the contexts when the foreground application changes.

1.8. Are there any Wintab features which are not supported by the Wacom Wintab?

Currently, the following Wintab features are not supported by the Wacom Wintab:

- The WTMgrDeviceConfig() function.
- The WTMgrConfigReplace() function.
- The WTMgrConfigReplaceEx() function.
- Packet hooks.
- The WTINFO_CHANGE message.
- The DVC_PNPID WTInfo index.
- Linking with WINTABX.LIB is not supported.
- Getting the packet rate with lcPktRate is not supported.

However applications can make use of packet timestamps - which are accurate - to determine packet rate. This can be done by setting the PK_TIME bit when defining PACKETDATA at compile time (see any of the demos for an example of how PACKETDATAis defined). The application then picks off the timestamp data in its WT_PACKET message handler.

Getting function key data with WTX_FKEYS is not supported.

2. Techniques

2.1. How can I tell if the user is using the tablet as the current pointing device or if she is using a mouse?

By processing the WT_PROXIMITY message you receive each time a tablet cursor enters or leaves the proximity of one of your context, you can tell if she is using the tablet. The low word of the lParam parameter of the WT_PROXIMITY message is nonzero when the cursor is entering your context and zero when the cursor is exiting your context. WT_PROXIMITY messages are also sent to your context when a cursor enters or leaves your context because your context's status changed. For example, if the pen is in proximity of the tablet when your context is opened, then you will receive a WT_PROXIMITY for the pen.

2.2. How do I tell when I am getting tablet data from a particular cursor, for example an airbrush?

In general, the steps are:

Tell Wintab you want to know which cursor generated each tablet packet.
Watch for the WT_CSRCHANGE message which is generated when a new cursor enters your context. When you receive a WT_CSRCHANGE message, one of the parameters is the packet serial number of the packet with the new cursor. Read or peek that packet to get the cursor number of the new cursor.

Use the cursor number to call WTInfo() to get detailed information about the new cursor to determine if it is the desired cursor. If this cursor number is the desired cursor, then you can be sure all packets in that context with this cursor number will be packets from the desired cursor until your context receives a WT_PROXIMITY leaving message.

A variation of this algorithm is to not use WT_CSRCHANGE or WT_PROXIMITY messages to watch for cursors entering and exiting and instead just watch the cursor numbers in the packets to see when a cursor enters or leaves your context.

In detail, the steps are:

1. Tell Wintab you want to know which cursor generated each tablet packet. Add the pkCursor field to the tablet packets your context receives by adding PK_CURSOR to the PACKETDATA symbol:

```c
#include <wintab.h>
#define PACKETDATA (PK_X | PK_Y | PK_Z | PK_BUTTONS | PK_NORMAL_PRESSURE |
                 PK_ORIENTATION | PK_CURSOR )
#define PACKETMODE 0
#include <pktdef.h>
```

and using the PACKETDATA for the value of the lcPktData field of your Wintab context:

```c
wWTInfoRetVal = WTInfo( WTI_DEFSYSCTX, 0, &logicalContext );

logicalContext.lcPktData = PACKETDATA;

hWintabContext = WTOpen( hWnd, &logicalContext, TRUE);
```

The pkCursor field of each tablet packet will now contain the cursor index of the cursor which generated the packet.

2. Watch for the WT_CSRCHANGE message which is generated when a new cursor enters your context. To get the WT_CSRCHANGE message, you must set the CXO_CSRMESSAGES bit of the lcOptions field of your logical context before you open it. Use the packet serial number in the wParam parameter of this message to peek or read the packet and get the value of the packet's pkCursor field, which contains the cursor number for the new packet.

```c
bRetVal = WTDataPeek( hWintabContext, lParam, lParam, 1, &packet, &nNumPacketsCopied );
```

3. Use the cursor number to call WTInfo() to get detailed information about the new cursor to determine if it is the desired cursor. Get the CSR_TYPE and CSR_PHYSID for the new cursor. The CSR_TYPE is a bitfield that identifies the type of cursor (pen, airbrush, etc). CSR_PHYSID is a serial number for the cursor. CSR_TYPE and CSR_PHYSID together comprise a completely unique id for the cursor, and are defined in Wintab.h. The file, DuoHand.h, in the CursorMask demo specifies bits that identify which cursors:

```c
#define CSR_TYPE_SPECIFIC_MASK ((UINT) 0x0F06)
#define CSR_TYPE_SPECIFIC_STYLUS_BITS ((UINT) 0x0802)
#define CSR_TYPE_SPECIFIC_AIRBRUSH_BITS ((UINT) 0x0902)
#define CSR_TYPE_SPECIFIC_4DMOUSE_BITS ((UINT) 0x0004)
#define CSR_TYPE_SPECIFIC_LENSCURSOR_BITS ((UINT) 0x0006)
```

And some code from the same CursorMask demo that uses these definitions:

```c
if ((pCtxInfo->wCursorType & CSR_TYPE_SPECIFIC_MASK) ==
    CSR_TYPE_SPECIFIC_4DMOUSE_BITS) { ... }
```

```c
wWTInfoRetVal = WTInfo( WTI_CURSORS + packet.pkCursor, CSR_TYPE, wCsrType );
wWTInfoRetVal = WTInfo( WTI_CURSORS + packet.pkCursor, CSR_PHYSID, wCsrPhysId );
```

// CSR_TYPE_SPECIFIC_MASK (0x0F06) is the bitmask used to find the specific cursor type.
if ((wCsrType & CSR_TYPE_SPECIFIC_MASK) == CSR_TYPE_SPECIFIC_AIRBRUSH_BITS) {
    //Cursor is an airbrush
}

As in all discussions in this FAQ, this example is for the first tablet on the system.

2.3. The Wintab SDK examples won't run without Wintab installed on the system. How can I make my program work if there is no tablet (no Wintab) on the system?

The demo examples at http://www.developer.wacom.com have been written to explicitly link to Wintab functions. When you implicitly link to Wintab (using static linking via a Wintab32.lib file), Windows resolves the references to Wintab functions for you when the user runs your program. If Wintab32.dll is not present on the system, the OS can't resolve the references and won't load your application.

To solve this, you should explicitly link to Wintab32.dll with your own code (Wintab32.dll is installed with any Wacom tablet driver). How to explicitly link to a DLL is described in detail in MSDN if you search titles for 'Link Explicitly'. You can also examine the Utils.c and Utils.h files in the demo examples.

Here is how you would explicitly link to WTInfo().

1. Define a function pointer for WTInfo.

   typedef UINT (API * WTINFO)(UINT, UINT, LPVOID);

2. Call LoadLibrary to load Wintab32.dll

   ghWintab = LoadLibrary("Wintab32.dll");
   if (!ghWintab)
   {
       return FALSE;
   }

3. Find the address of WTInfo(). Wintab functions that return strings have both an ASCII and UNICODE version. We want the ASCII version.

   WTINFO pWTInfo;
   pWTInfo = (WTINFO)GetProcAddress(ghWintab,"WTInfoA");
   if (!pWTInfo)
   {
       return FALSE;
   }

Now you can use the function pointer to call WTInfo

   wWTInfoRetVal = pWTInfo(WTI_INTERFACE,IFC_SPECVERSION,&wSpecVersion);

4. Unload Wintab32.dll when your program quits.

   if (ghWintab)
   {
       FreeLibrary(ghWintab);
   }

2.4. How do I get data for the airbrush fingerwheel?

The airbrush fingerwheel data is returned as tangent pressure. The data is returned in the pkTangentPressure field of the tablet data packets. You can tell Wintab to add tangent pressure to your packets by adding PK_TANGENTPRESSURE as one of the data items to your PACKETDATA definition. The range of values which will be returned by tangent pressure can be obtained by calling WTInfo( WTI_DEVICES, DVC_TPRESSURE, &axis );

2.5. How do I support the Intuos tablet DualTrack feature, which allows two-handed input from the user?

One way to support two-handed input is to use the Wintab CSRMASK extension and two Wintab contexts, one which moves the system pointer and one which does not. See the CursorMaskEx example from http://www.developer.wacom.com for an example of how to use these Wintab features to support two handed input.
Note that DualTrack was only supported for the following tablets: UDII 12x12, Intuos and Intuos 2. None of the Intuos 3, Intuos 4, Cintiq or Bamboo lines support Dual Track.

2.6. How can I improve the performance of my tablet-aware code?

Keep in mind that Wintab function calls are expensive. To improve performance of your application, minimize the number of Wintab calls you make, especially the number of packet functions such as WTPacket() and WTPacketsGet(). The best strategy is to use WTPacketsGet() to retrieve the maximum number of packets your context queue can hold. The return value of WTPacketsGet() is the number of packets returned. You can find out the size of your queue by calling WTQueueSizeGet(). It is not necessary and very expensive to call WTQueuePacketsEx() to discover the number of packets waiting in your queue.

When you call Wintab functions there is a task switch. YourWndProc can easily go reentrant if you do much tablet processing in it. You might want to test for reentrancy in your code and synchronize access to global or shared resources which would be effected by reentrancy.

2.7. How can I detect transducer button presses?

The CadTest demo is one example that shows how to detect when any button is down using the PACKET::pkButtons member. When the WT_PACKET message is received, you can check for a button press such as this:

```c
    case WT_PACKET:
        if (gpWTPacket((HCTX)lParam, wParam, &pkt))
        {
            if (HIWORD(pkt.pkButtons)==TBN_DOWN)
            {
                // ...
            }
        }

        // ...
        break;
```

Note that to get button data in the packet information you must enable button data in the PACKETDATA definition, such as the following:

```c
#define PACKETDATA PK_X | PK_Y | PK_BUTTONS
```

3. Common Problems

3.1. Why am I only getting Z coordinates of zero from the thumbwheel of the 4D mouse?

The thumbwheel of the 4D mouse can have several functions. The user sets the function of the thumbwheel in the Thumbwheel tab of the Wacom Intuos control panel applet. The default function of the thumbwheel is Intellimouse emulation. To receive Z coordinates from the thumbwheel, the user must go to the thumbwheel tab and set the function to ‘Application Defined’. There is no way to change the function from your program.

3.2. Why doesn't the screen cursor move when my Wintab program is running and the user is using the tablet?

The CXO_SYSTEM bit of the lcOptions field of your logical context controls whether Wintab moves the system pointer (screen cursor) when your context receives tablet data. If this bit is not set, then Wintab will not move the system cursor when your application is active. If you use WTInfo( WTI_DEFCONTEXT, … ) to get context information with which to create your own context, CXO_SYSTEM will be clear unless you set it before calling WTOpen().

```c
    LOGCONTEXT wintabLogicalContext;

    wWTInfoRetVal = WTInfo( WTI_DEFSYSCTX, 0, &wintabLogicalContext );

    wintabLogicalContext.lcOptions |= CXO_SYSTEM;
```
3.3. Why do I suddenly stop receiving WT_PACKET messages from the tablet or seem to only get some of the tablet packets?

You are probably missing tablet packets because your context's packet queue is overflowing. Each Wintab context has a queue of tablet packets. Wintab only adds new packets to the queue and sends your application a WT_PACKET message if the queue is not full. If the queue is full, Wintab discards the packet and does not send your application a WT_PACKET message to tell you that your context received a tablet packet. The default queue size is only eight packets and so overflows very quickly since the Intuos tablet sends between 100 and 200 packets a second.

To prevent your queue from overflowing, increase the size of your context's queue with the WTQueueSizeSet() function to a value between 32 and 128. Memory for packet queues is a limited resource, so be sure check that your call to WTQueueSizeSet() succeeds. If WTQueueSizeSet() fails, request a smaller queue size.

You can also help prevent the queue from overflowing by always removing packets from the queue when you receive a WT_PACKET message and by flushing the queue when you receive WT_PROXIMITY messages. You can flush the queue by calling:

\[
\text{WTPacketsGet( hWintabContext, VALUE_LARGER_THAN_YOUR_QUEUE_SIZE, NULL);}
\]

or

\[
\text{WTPacket( hWintabContext, wPacketSerialNumber, NULL );}
\]

3.4. Why does my application sometimes not receive tablet data if another tablet-aware (Wintab) application is running?

Your application is not receiving tablet data because it is not at the top of the overlap order of Wintab contexts. Wintab contexts are stacked upon one another. In general, only the topmost context in the overlap order receives tablet data.

When an application becomes the active application, it should move its context to the top of the overlap order by calling WTOverlap( hWintabContext, TRUE ). When an application becomes inactive, it should call WTOverlap( hWintabContext, FALSE) to move its context to the bottom of the overlap order. The CursorMask example shows how to appropriately handle context overlap order as a result of a WM_ACTIVATE message.

The Wacom implementation of the Wintab specification differs in a couple of ways. The primary difference is in how Wacom handles the contexts created by an application. The Wintab specification requires an application to handle the context z-order manually. That is to say that when an application context is created it will be placed on top of all other context and it will stay on top unless the context is destroyed, a new context is created or the WTOverlap function is called to move the context to the bottom. This means that applications should move their context to the bottom of the list when they are minimized and then move the context back to the top when they are restored. During development of the Wacom Wintab driver, we discovered that many applications do not manage the contexts correctly. Wacom decided to manage to context order for the applications.

Wacom automatically moves a context to bottom of the stack when the application loses focus and moves the context to the top when the application gains focus. This solves a lot of problems related to multiple Wintab application active on a system at the same time. The side affect is that if you are creating an application that will collect tablet data for a second application, because your application does not have the system focus, your application will fail to get any tablet data.

3.5. How can I tell if an attached tablet is a Wacom tablet?

There is currently no way to tell if an attached tablet is truly a Wacom tablet. Some other tablet manufacturers report their tablets as 'Wacom' tablets when you call WTInfo( WTI_DEVICES, DVC_NAME, ... ).

However, using the WTInfo() function, you can find what data items are supported by the tablet and what their ranges will be.

3.6. How can I tell which Wacom tablet model is being used?

There is currently no way to tell which tablet model is installed on the machine. However, using the WTInfo() function, you can find what data items are supported by the tablet and what their ranges will be.

3.7. If the user uses the Wacom control panel applet to customize my application or a tool (for example, sets only a portion of tablet active), the customized setting always effects my application. Is this a bug, by design, and can I stop it?

This is by design. One of the design goals of Intuos was to give the user maximum flexibility and control over the behavior of the tablet. Part of this control is application and cursor (tool) specific settings. There is, of course, a tension between allowing the user to customize settings for applications or tools and application code needing to set settings so that their applications work as the application vendors intended.
Graphics tablet applications can be generally divided into two categories: Applications which need tablet coordinates mapped to the physical coordinate space they request and those applications which don't. CAD applications, digitizing applications, and template utilities are examples of the first kind of application. Painting and drawing programs are examples of the second kind of application. There are many more applications in the second category than in the first.

In most cases, the Wacom Wintab lets the user's custom settings override the settings the application makes using the Wintab API. However, to accommodate CAD and other applications in the second category, the Wacom Wintab will let the application's Wintab context settings override the user's custom settings if the application opens a Wintab context with any of these characteristics:

- The context has any bits set in lcLocks.
- The application changes the tablet input origin (lcInOrgX, lcInOrgY or lcInOrgZ).
- The application changes the tablet input size (extents) (lcInExtX, lcInExtY or lcInExtZ).
- The application changes the screen origin (lcSysOrgX or lcSysOrgY).
- The application changes the screen output size (extents) (lcSysOrgX or lcSysOrgY).
- The application sets the screen cursor to move in relative mode (sets lcSysMode to non-zero).

3.8. The lens cursor and 4D mouse always move the screen pointer in relative (mouse) mode rather than pen (absolute) mode. How can I change this?

This question is a variation of FAQ 3.7. Please see that FAQ for an answer.

3.9. Why doesn't the lcPktRate field of my logical context show the correct report rate for the tablet?

The lcPktRate field is not functional in the non-Wacom Wintab which shipped with older Wacom tablets. The Wacom Wintab currently doesn't support lcPktRate either. In both Wintabs, lcPktRate is reported as 100, but the tablet will actually be sending data at the highest rate possible for whatever mode in which the tablet is set.

3.10. How can I use the .NET InkCollector object to obtain pen data from my tablet?

If you are programming for Vista or Windows 7, there are a number of examples showing how to use InkCollector (which cannot be used on Windows XP). In addition to referring to the .NET documentation on msdn.microsoft.com you can see these web pages for TabletPC inking examples:

http://msdn.microsoft.com/en-us/magazine/cc164069.aspx (this one includes pressure examples)

3.11. How can I capture pen data into a LabView application?

LabView can be integrated with DLLs, so you can use the Wintab API (supported by Wintab32.dll) to provide tablet pen data (location and pressure). There is a book that describes how to integrate LabView and DLLs ("LabView advanced programming techniques", by Rick Bitter, Taqi Mohiuddin, Matt Nawrocki)

A paper detailing how to do this can also be found at: ftp://volkar.goldns.com/CAN%20Controller%20Area%20Network/epos2/04%20Programming/Windows%2032-Bit%20DLL/LabVIEW/Documentation/EPOS%20DLL%20Integration%20into%20LabView.PDF

The .NET InkCollector API can also be used to capture pen data.

3.12. Why does the Win32 function, GetRawInputData() not work with a Wacom tablet?

This function does not work because it cannot read the tablet driver's custom pen data collections.

3.13. How can I use tablet data with web applications?

You can use the Wacom webplugins to extract pen location, pressure, rotation, orientation, and tilt data. The Wacom webplugins are installed on the user's system (Windows and Mac) when the tablet driver software is installed. There are example web pages that use the plugin at http://www.developer.wacom.com.
3.14. Is there an API for Bamboo touch data?

There is an API for Wacom feel™ Multi-Touch Data.

Check out the list of currently supported touch-enabled tablets in Wacom feel™ Multi-touch FAQ

3.15. Is there a Wintab .NET API?

There is now a .NET API for Wintab. You can access the API, which includes documentation and sample code at: WintabDN.zip

3.16. How can I access Bamboo pen and touch data in a WPF application?

You can use the WPF InkCanvas object to capture pen data. There is a sample application at http://www.codeproject.com/KB/WPF/Ink1.aspx that should get you started with pen data capture. As to touch data, InkCanvas will not recognized Bamboo touch data.

3.17. How can I get Intuos altitude, azimuth and twist data?

The standard grip pen should give you altitude and azimuth. The TILTTEST demo from our developer site shows an example of how to do this. You will only get twist data if you use an Art pen, which can be verified from the same app.

3.18. How can I map tablet area to application pixel area?

Here is one way to map tablet to pixels (taken from the MfcDemo Wintab app). Here are some code snippets that point out the salient ideas. Basically we're just doing interpolation between the pixels using the higher resolution tablet coordinates.

```c
CMFC_DEMOView::OnCreate
{
    // Get default context information
    // Be sure to use default digital context for data packets before
    // calling gpWTOpenA.
    gpWTInfoA( WTI_DEFCONTEXT, 0, &lc );
    ...
    gpWTOpenA( ... );
    ...
}

CMFC_DEMOView::OnWTPacket
{
    ...
    POINT csr;
    ...

    // Read the packet to get tablet coordinates.
    PACKET pkt;
    gpWTPacket( (HCTX)hCtx, wSerial, &pkt );
    ...

    // Get window size to get pixel extents.
    RECT window_rect;
    GetWindowRect( &window_rect );
    POINT size;
```
3.19. WPF’s InkCanvas does not report multitouch events in Windows 7 (on some TabletPCs).

This is due to the Wacom touch driver not supporting the device ID string descriptor. Microsoft has provided a fix (see http://support.microsoft.com/kb/2328751). It only applies to .NET 4.0 and greater. Here is a link to where you can currently download the fix: http://code.msdn.microsoft.com/KB2328751/Release/ProjectReleases.aspx?ReleaseId=5030

3.20. How can I tell if the webplugin is installed and working for my browser?

For the webplugin to work, the latest tablet driver (consumer or professional) must be installed. This will install the necessary webplugin installer as well as the Wintab supporting files needed for the Webplugin to communicate with the tablet driver. Once you’ve installed the driver, just open one of the webplugin demos and try it out (for example, try the Scribble demo).

In Firefox, you can check to see whether the plugin is installed by typing “about:plugins” in the URL field. You should see an entry for “Wacom Dynamic Link Library”. This also works in Chrome.

In Internet Explorer, click on Tools | Manage Add-ons, and then select from the drop-down Currently loaded add-ons | All Add-ons. You should see an entry for WacomTablet Class.

3.21. Where can I find out about Linux tablet programming?

Visit the Linux Wacom Project or OpenSolaris Archive for support.

3.22. How can I find out how many (if any) tablets are plugged in?

WTInfo(WTI_INTERFACE, IFC_NDEVICES, &numDevices) will return the number of connected tablets or zero.

3.23. How can I program an STU-300, STU-500, or STU-520 Signature (signpad) Tablet?

Here is where you can obtain information about Wacom’s Signature Solutions.

3.24. Is there MATLAB support for Wintab?

No, we currently do not have integrated MATLAB support. Such support has been created, so it is possible (see: “PENTOOLS - A MATLAB Toolkit for On-Line Pen-based Data Experimentation”, by R. M. Guest - http://www.cvc.uab.es/icdar2009/papers/3725b221.pdf).

3.25. Does Wintab support WTX_TILT?

No, our implementation of Wintab does not support the WTX_TILT extension. However, the standard grip pen should give you altitude and azimuth, from which you can calculate tilt (see the TILTTEST demo).

3.26. Why can’t I get the full tablet extended range of values?

The tablets are manufactured with a small portion of the tablet active area under a margin around the drawing area.

3.27. Is it possible to use Wintab to identify which model tablets are connected?

The short answer is no. The longer answer is that Wintab provides the ability to query (through WTInfo), the capabilities of the connected tablet. Wacom has determined that this is the better way to keep your application robust (as opposed to relying on model numbers that can go out of date).

3.28. What tool can I use to determine if a tablet is sending messages?
The hclient tool, in the WinDDK, is a good way to determine if your tablet is sending pen messages. This tool does not rely on Wintab; rather it connects to a tablet's HID descriptor, and so can monitor tablet data as it is sent from the HID kernel stack. Simply select the device from the list of HID devices that hclient has enumerated (note that the Vendor ID = 0x56a, which is for Wacom). Make sure that the Usage Page = 0xD, and the Usage = 0x1. Press the "Blocked Read Data" button at the bottom of the dialog, and press the "Continuous Asynchronous Read" button when the data dialog comes up. Touch the pen to the tablet to see the data.

3.29. Why can't I find the old Wintab32x.lib library?

This older library requires static linking, and as such can cause a program to break if that program is executed on a system with a newer version of Wintab32. Thus we strongly encourage dynamically loading the Wintab32.dll within a client application. The older library, if you still need it, is available upon request.

3.30. How can my application have exclusive access to all tablet pen events?

You must force your application to be the front-most (foreground) app.

3.31. How can I get low-level interface documentation for Wacom tablets?

The short answer is that you normally can't because this is proprietary information that can only be released under an NDA. The long answer is that you should not need such information because Wintab supports obtaining tablet pen data (or you can use .NET InkCollector on Vista and Windows 7 systems).

3.32. Why do I get the error: "Could not load wintab32.dll" when I try to run the Wintab demos?

Normally, this means that your application could not get information from WTInfo(), and this is usually because of one of the following reasons:

- The tablet driver is not installed, or the installation has been corrupted. Try reinstalling the driver.
- The tablet driver is not communicating through Wintab. Try rebooting the system.

3.33. How can I determine if a key combination comes from a pen or tablet button as opposed to the keyboard?

The tablet driver software attempts to emulate a keyboard key combination press as closely as possible by sending the key combinations as would be sent by the keyboard. To determine whether a key combination comes from the pen or tablet, you can set up the tablet preferences to send a key combination that contains some extra character, which your application can detect and strip out. For example, you can set up the tablet preferences to send "Ctrl + Z + t" for a "Ctrl + Z" combination, and have your app strip out the "t". If the "t" is not there, you know it came from the keyboard.

3.34. What Wintab function do I use to retrieve PACKETEXT data?

If your application has set up to retrieve PACKETEXT data, use WTPacket() to query for data in response to WT_PACKETEXT messages.

3.35. Can my application retrieve a tablet-to-screen user preferences mapping through Wintab?

The short answer is no. The longer answer is that Wintab goes to great lengths to hide that mapping, which allows an application to not have to deal with tablet-to-screen mappings for preferences set outside of an application's context. Wintab supports letting an application set its tablet-to-screen mapping anyway if it wants - which overrides the preferences settings.