NextWindow

USB

Application Programmer Interface

User Guide

This document describes NextWindow's Touch Application Program Interface (API) Beta version. This software is under development and is currently provided for testing purposes only. Please contact NextWindow for the latest release information.

Revision History

User Guide v1.2 27 March 2008
ghostTouch parameter added to the getpolygon() function

User Guide v1.3 16 June 2008
Touch confidence level explained
GetConnectedDisplayCount ( ) added
GetConnectedDisplayInfo( ) added
Error codes added:
- ERR_DEVICE_DOES_NOT_EXIST
- ERR_DISPLAY_DOES_NOT_EXIST
- ERR_FUNCTION_NOT_SUPPORTED

NWDisplayInfo structure added
rect_t structure added
deviceID argument added to:
- SetTouchScreenDimensions ( )
- GetTouch ( )
- GetCentroid ( )
- GetPolygon ( )

User Guide v1.4 07 July 2008
Slopes mode for firmware 2.98 added
Error codes added:
- ERR_INVALID_SENSOR_NUMBER
- ERR_SLOPES_MODE_NOT_SUPPORTED

SetNearFieldRegion ( ) added
GetNearFieldRegion ( ) added

API v1.0.1 16 June 2008
Improved 2-Touch Reliability
Touch Confidence Reporting
Support for Multiple Touch Devices
Various Bug Fixes
Ability to retrieve information about connected displays
Updated sample code – C++ and C#

API v1.1 July 2008
Supports “slopes mode” for firmware v2.98 and later
Improved two-touch reliability
Various bug fixes

API v1.2
Bug fixes

API v1.3 and User Guide v1.5 September 2008
Improved two-touch reliability
User Guide, update API version number

API v 1.2.0.0
Added gesture support
GetCentroid rewritten to ignore off-screen touches.
Improved two-touch reliability.
Various bug fixes.
Updated sample code for C++ and C#
Fixed memory leaks.

User Guide version 1.6 November 2008
NextWindow Gestures section added
Connected Devices and Disconnected Devices added

API v 1.2.6 and User Guide v 1.7 15th April 2009

Improvements in picking correct touch pair when using RM_SLOPESMODE.
Fixed bug with GetReportMode sometimes not returning correct report mode.
Added compatibility with 1900 Windows 7 driver.
Fixed bug with width/height/area not being reported correctly.

Table of Contents

Introduction

NextWindow's touch Application Program Interface (API) provides programmers with access to the raw touch data generated by a NextWindow touch screen. It also provides derived touch information such as velocity and acceleration of the touch point.

The touch events, data and derived information can be used in any way the application wants.

Communications are via HID-compliant USB.

Multi-Touch

Multi-touch simply refers to a touch-sensitive device that can independently detect and optionally resolve the position of two or more touches on screen at the same time. In contrast, a traditional touch screen senses the position of a single touch and hence is not a multi-touch device.

- “Detect” refers to the ability to sense that a touch has occurred somewhere on screen.
- “Resolve” refers to the ability to report the coordinates of the touch position.

NextWindow’s standard optical touch hardware can detect two touches on screen. The firmware can resolve the position of two simultaneous touches (with some limitations, which are explained in this guide).

NextWindow Touch API Download

Visit http://www.nextwindow.com/support/application_notes/api.html

Touch Event Data

Multi-Touch Data

The optical sensors detect the position of the finger (or stylus) as it touches the screen.

The firmware reports the following data for each of the two touch positions:

- Touch event and type (touch down, touching, or lift off).
- Continuously updated X and Y coordinates during touch.
- Width of touch as seen by the sensors.

Derived Information

In addition to gathering raw data, in mouse mode (see below), the software calculates and provides the following for each of the two touch positions:

- Duration of touch.
- Time between two touches.
- Size of touch area.
- Direction of travel (if finger moving across the surface).
- Velocity of travel.
- Acceleration of travel.

NextWindow's firmware versions prior to 2.98 can be set to operate in three different modes: mouse mode (the default), multi-touch mode and digitiser mode. Firmware versions 2.98 onwards include slopes mode.

Mouse Mode

In mouse mode, the firmware converts touch events to regular mouse operations, including left click, drag, right click and double click.

Multi-Touch Mode

In multi-touch mode, touches are sent through the USB HID communications channel as raw data (touch event, XY coordinates and touch width).

Digitiser Mode

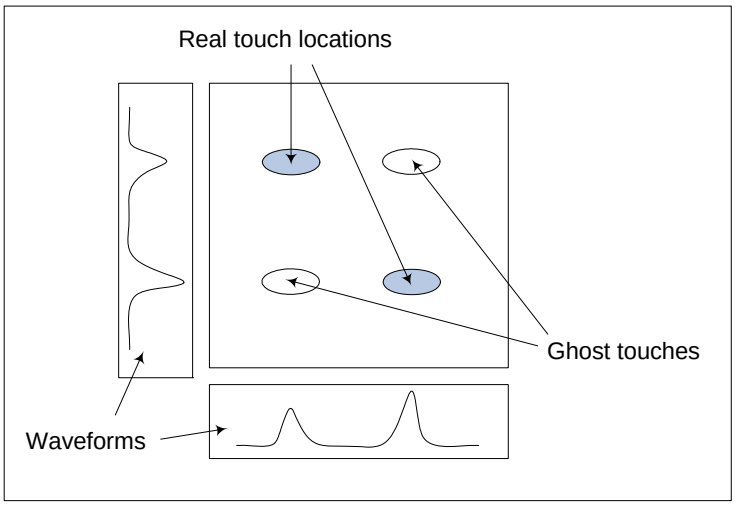
In digitiser mode, touch events are passed directly through the touch screen's HID digitiser endpoint. Processing is done by the application or digitiser driver.

Slopes Mode (Firmware v2.98 onwards)

In slopes mode, touch events are reported via USB as slope data rather than coordinates and the DLL converts the data. The result is more accuracy for two touch.

Limitations

- Touches should touch down 20 ms or more apart for maximum two-touch resolvability
- In most multi-touch systems, when more than one touch is present, there are two possible positions for each touch. This is illustrated below for a rectangular coordinate system.



The real touch locations can be resolved in software. The other two possibilities are called “ghost touches.”

In many operations (such as rotating and stretching a shape), ghost touches are just as useful as regular touches in the computation.

- When moving touch points leave an occluded state (two touches in line with one of the optical sensors), there can be further ambiguity between the real and the ghost touches. This can be resolved in software to a reasonable degree of certainty.

Data Structures

The following data structures should be used when interfacing with the API.

Touch Point Structure

The individual touch point information for each of the touches, ghost touches and centroid are stored in a structure, NWTouchPoint. This is populated by the GetTouch( ) and GetCentroid( ) functions.

```
typedef struct
  {
  Dword       touchID;
  Dword       touchType;
  __int64     touchStart;
  Point_t     touchPos;
  Float       velocity;
  Float       acceleration;
  Float       touchArea;
  Dword       touchEventType;
  Dword       confidenceLevel;
  Dword       height;
  Dword       width;
  } NWTouchPoint;
```

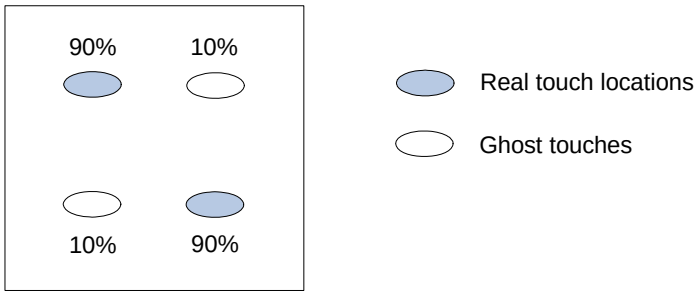
The members of NWTouchPoint are explained in Table 1 below.

Member Variable	Description
touchID	A unique ID to identify the touch for tracking purposes.
touchType	The status of the touch – this has three possible values: TT_TOUCH TT_GHOSTTOUCH TT_CENTROID.
touchStart	A timestamp value to indicate the time at which the touch originated.
touchPos	A point structure containing X and Y coordinates for the touch.
velocity	Reserved for velocity (future release)
acceleration	Reserved for acceleration (future release)
touchArea	The area covered by the touch.
touchEventType	The status of the touch – this has three possible values : TOUCH_DOWN – Indicates that the touch has just been generated. TOUCHING – Indicates that an existing touch is still being tracked. TOUCH_UP – Indicates that the touch has been removed.
confidenceLevel	An integer value between 0 and 100 providing the level of certainty that the touch point is a real touch. A value of 0 indicates no certainty and a value of 100 indicates absolute certainty. See Touch Confidence Level on page 7.
height	A value between 0 and 1 representing the height of the touch.
width	A value between 0 and 1 representing the width of the touch.

Table 1 – Members of NWTouchPoint

Touch Confidence Level

The confidenceLevel reports the likelihood that a pair of touches are the true touches. In the diagram below, the pair of touches with 90% confidence level will be interpreted as the true touches, and those with 10% will be interpreted as the ghost touches.



The sum of the confidences between true and ghost levels add up to 100%. At 50:50, there is no certainty about which is which. The confidenceLevel values are reported at each packet. If it becomes clear, due to the confidence level rising, that a pair of ghost touches are actually the true touches, the reported ghost touches / real touches do not suddenly flip over; they remain tracking until one or both of the touches are lifted.

Touch Device Information

Use the GetTouchDeviceInfo( ) function to retrieve details of a particular touch device. This data is returned in a NWDeviceInfo structure as shown below.

```
typedef struct
{
Dword    serialNumber;
Dword    modelNumber;
Dword    firmwareVersionMajor;
Dword    firmwareVersionMinor;
Dword    productID;
Dword    vendorID;
} NWDeviceInfo;
```

The members of NWDeviceInfo are explained in Table 2.

Member Variable	Description
deviceID	A 32 bit value to uniquely identify the touch device.
serialNumber	The serial number of the touch device.
modelNumber	The model number of the touch device.
firmwareVersionMajor	A 32 bit value to identify the version of the firmware that is running on the touch device.
firmwareVersionMinor	A 32 bit value to identify the version of the firmware that is running on the touch device.
productID	ProductID value of the touch device.
vendorID	VendorID value of the touch device.

Table 2 – Members of NWDeviceInfo

Display Screen Information

Use the GetConnectedDisplayInfo( ) function to retrieve details of the display connected to the operating system. This data is returned in a NWDisplayInfo structure as shown below.

```
typedef struct
{
  DWORD   deviceNo;
  rect_t  displayRect;
  rect_t  displayWorkRect;
  bool    isPrimary;
  char    deviceName[CCHDEVICENAME];
} NWDisplayInfo;
```

The members of NWDeviceInfo are explained in Table 3.

Member Variable	Description
deviceNo	The display index (as passed to GetConnectedDisplayInfo)
displayRect	A rect_t structure that specifies the dimensions of the display monitor, expressed in virtual-screen coordinates. Note that, if the monitor is not the primary display monitor, some of the coordinates may be negative.
displayWorkRect	A rect_t structure that specifies the work area rectangle of the display monitor, expressed in virtual-screen coordinates. Note that if the monitor is not the primary display monitor, some of the coordinates may be negative.
isPrimary	Whether the display is the primary display.
deviceName	the name of the display.

Table 3 – Members of NWDisplayInfo

Display Screen Coordinates

```
typedef struct
{
  float left;
  float top;
  float right;
  float bottom;
} rect_t;
```

left and **top** specify the X and Y coordinates for the top left corner of the rectangle.
right and **bottom** specify the X and Y coordinates for the bottom right corner of the rectangle.

Function Return Values

Many of the functions return a success code. Possible values are shown in Table 4.

Value	Name	Description
1	SUCCESS	Success.
-1	ERR_DEVICE_NOT_OPEN	Operation failed – Device not open. The device has not been initialized.
-2	ERR_INVALID_PACKET_ID	Operation failed – Invalid packet ID. The referenced packet does not exist.
-3	ERR_INVALID_TOUCH_ID	Operation failed – Invalid touch ID. The touch does not exist for the referenced packet.
-4	ERR_TOO_MANY_TOUCHES	Operation failed – Too many touches. Too many touches were specified for the function in question.
-5	ERR_DEVICE_DOES_NOT_EXIST	Operation Failed – The referenced device does not exist.
-6	ERR_DISPLAY_DOES_NOT_EXIST	Operation Failed – The referenced display does not exist
-7	ERR_FUNCTION_NOT_SUPPORTED	Operation Failed – The function is not supported with this version of the firmware.
-8	ERR_INVALID_SENSOR_NUMBER	Operation Failed – The referenced sensor does not exist
-9	ERR_SLOPES_MODE_NOT_SUPPORTED	Operation Failed – Slopes mode is not supported for firmware versions earlier than 2.98

Table 3 – Function return values

Receiving Data Packets

Event Handler

TouchPacketReceived ( )

Receives touch data packets.

```
void TouchPacketReceived (Dword deviceID, Dword deviceStatus, Dword packetID,
                          Dword touches, Dword ghostTouches);
```

Argument	Type	Description
deviceID	Dword	
deviceStatus	Dword	DS_CONNECTED – sent when device connected DS_TOUCHINFO – has sent valid touch information DS_DISCONNECTED – sent when device disconnected
packetID	Dword	ID representing the current packet being reported. 255 packets are stored at a time.
touches	Dword	32 bit value that contains a list of the touch IDs that are present for this particular touch packet
ghostTouches	Dword	Contains the list of the touch IDs for each of the ghost touches

Table 4 – arguments for TouchPacketReceived function

Storage of Touch IDs

The touch IDs are stored in a 32 bit variable with each bit referencing a different touch ID.

For example:

```
00000000000000000000000000011101
```

This 32 bit value contains the touch IDs : 1, 4, 8 and 16.

Connected Devices

To receive notification when a device is plugged into the system, the application should include an event handler with the following prototype:

```
void DeviceConnectHandler (DWORD deviceID)
```

The application then calls the following function with a pointer to the event handler:

```
void SetConnectEventHandler (NWConnectEvent cHandler)
```

Note: On application shutdown, call SetConnectEventHandler(NULL) to prevent memory leaks.

Disconnected Devices

To receive notification when a device has been unplugged from the system, the application should include an event handler with the following prototype:

```
void DeviceDisconnectHandler (DWORD deviceID)
```

The application then calls the following function with a pointer to the event handler:

```
void SetDisconnectEventHandler (NWDisconnectEvent dHandler)
```

Note : On application shutdown, call SetDisconnectEventHandler(NULL) to prevent memory leaks.

Initializing and Closing the Device

OpenDevice ( )

Starts communications between the DLL and the touch screen and between the application program and the DLL.

```
Dword OpenDevice (Dword deviceID, NWTouchPacketCallback* callbackFn);
```

Returns SUCCESS or an error code (see Table 3 on page 9).

The callback function specifies the event handler that will be called whenever there are touch packets ready to send.

Note: On application shutdown, any open devices should be closed using the CloseDevice( ) function to prevent memory leaks.

GetConnectedDeviceCount ( )

```
Dword GetConnectedDeviceCount ( )
```

Returns the number of connected devices available for touch communications.

GetConnectedDeviceID ( )

```
Dword GetConnectedDeviceID (Dword deviceNo)
```

Returns the serial number of the specified device.

deviceNo specifies the number of the device to query.

GetTouchDeviceInfo ( )

Returns information about the touch device.

```
NWSuccessCode GetTouchDeviceInfo (Dword deviceID, NWDeviceInfo* deviceInfo);
```

The information is returned in deviceInfo (see NWDeviceInfo structure on page 7).

GetConnectedDisplayCount ( )

```
DWORD GetConnectedDisplayCount ( )
```

From operating system data, gets the number of display screens connected.

GetConnectedDisplayInfo ( )

Takes the display index and populates the NWDisplayInfo structure.

```
DWORD GetConnectedDisplayInfo(int displayNo, NWDisplayInfo* displayInfo)
```

The information is returned in displayInfo (see NWDisplayInfo structure on page 8).

SetTouchScreenDimensions ( )

```
void SetTouchScreenDimensions (Dword deviceID, float xMin, float yMin,
                                  float xMax, float yMax);
```

Used to set the physical size of the touch screen in units of the user's choice. All touch data will then be reported in these units.

The software automatically detects the size of the connected touch screen and sets the defaults as follows:

xMin = 0
yMin = 0
xMax = width of the connected touch screen in pixels
yMax = height of the connected touch screen in pixels

CloseDevice ( )

Close the device and stops receiving touch packets.

```
Void CloseDevice (Dword deviceID);
```

deviceID specifies the serial number of the device to close.

Retrieving Touch Data

GetTouch ( )

Retrieves the data for a particular touch.

```
NWSuccessCode GetTouch (Dword deviceID, Dword packetID,
                NWTouchPoint* touchData, Dword touch, Dword ghostTouch);
```

The application can either pass in a valid touch ID (and set the ghostTouch parameter to zero) or pass in a valid ghostTouch ID (and set the touch parameter to zero). Failure to do this will result in an error code being passed back (ERR_INVALID_TOUCH_ID).

GetCentroid ( )

Calculates the centroid of any number of existing touches and ghost touches.

```
NWSuccessCode GetCentroid (Dword deviceID, Dword PacketID,
           NWTouchPoint* touchData, Dword touches, Dword ghostTouches);
```

Calculates the centroid of the touches specified in touches and ghostTouches. If any of these touches are invalid, the function will fail, returning error code ERR_INVALID_TOUCH_ID.

GetPolygon ( )

Calculates the shape of the touch point.

```
Dword GetPolygon (Dword deviceID, Dword packetID, Dword touch,
                  Dword ghostTouch, Point_t* pointArray, Dword size);
```

GetPolygon( ) can be used in one of two ways:

First Method

- Call the function passing null and 0 as the pointArray and size parameters respectively, in order to return the number of points that make up the polygon for the touch for the specified packet.
- Allocate memory for the point array based on the number of points returned.
- Call the function, passing in a pointer to the point array and the number of points.

Second Method

- Create an arbitrary size array of points.
- Call the function, passing in a pointer to the array and the size of the array as the pointArray and size parameters respectively. The function will return the number of points that were actually copied into the array.

Touch Measurements

GetAngleBetweenPointsDegrees ( )

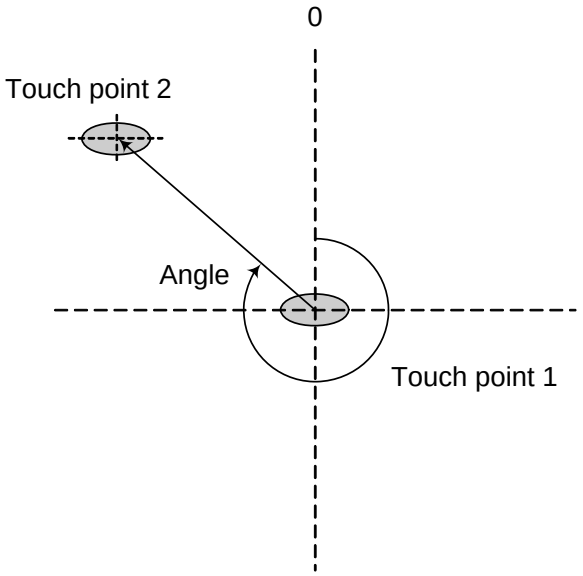
```
Float GetAngleBetweenPointsDegrees (Point point1, Point point2);
```

Returns the angle of touch point 2 relative to touch point 1 in degrees.

GetAngleBetweenPointsRadians ( )

```
Float GetAngleBetweenPointsRadians (Point point1, Point point2);
```

Returns the angle of touch point 2 relative to touch point 1 in radians.



GetDistanceBetweenPoints ( )

Retrieves the distance between two touch points.

```
Float GetDistanceBetweenPoints (Point point1, Point Point2);
```

The following two functions convert angles from degrees to radians and vice versa.

ConvertRadiansToDegrees ( )

```
Float ConvertRadiansToDegrees (float radians);
```

ConvertDegreesToRadians ( )

```
Float ConvertDegreesToRadians (float degrees);
```

Control Functions

Kalman Filter Control functions

Kalman filters are used to reduce noise and smooth the data.

SetKalmanFilterCoefficients ( )

Sets the Kalman filter coefficients.

```
void SetKalmanFilterCoefficients(DWORD deviceID, float g, float h, float k);
```

SetKalmanFilterStatus ( )

```
void SetKalmanFilterStatus (DWORD deviceID, bool kalmanOn);
```

Turns the Kalman filter on or off. Default is off.

Report Mode functions

The following flags can be used to set the report modes of the device:

RM_NONE

RM_MOUSE

RM_MULTITOUCH

RM_DIGITISER

RM_SLOPESMODE (Firmware v2.98 onwards)

SetReportMode ( )

Sets the report mode of the touch device (NextWindow firmware version 2.91 and later)

Returns SUCCESS or an error code (see Table 3 on page 9).

Example

To set both mouse and multi-touch, you would use :

```
reportMode = RM_MOUSE | RM_MULTITOUCH;
DWORD SetReportMode (DWORD deviceID, DWORD reportMode);
```

GetReportMode ( )

Gets the current report mode of the device. (NextWindow firmware version 2.91 and later)

```
DWORD GetReportMode (DWORD deviceID);
```

SetNearFieldRegion ( )

Relevant to firmware v2.98 onwards.

Sets the size of the nearfield regions. See *Nearfield Regions* below.

```
DWORD SetNearFieldRegion (DWORD deviceID, Point_t nearfield )
```

GetNearFieldRegion ( )

Relevant to firmware v2.98 onwards.

Gets the size of the nearfield regions. See *Nearfield Regions* below.

```
DWORD GetNearFieldRegion (DWORD deviceID, Point_t* nearfield )
```

Nearfield Regions

In positions close to the sensors, it is possible for reflections to cause the sensors to report multiple touches where there is in fact only one. Setting a nearfield region defines an area where the DLL reports multiple touches as one in order to avoid this effect.

The DLL sets default nearfield regions and normally there would be no need to take any further action. However, you can use GetNearFieldRegion( ) to read the values or SetNearFieldRegion( ) to change them if required.

The two nearfield regions are symmetrical (defined by one set of x, y positive values).

NextWindow Gestures

NextWindow touch screens allow two-touch gestures to be used where supported by the application.

You perform a gesture by double-tapping or dragging two fingers on the touch surface.

Types of Gestures

```
MTG_NONE = 0,        // No gesture (invalid)
MTG_TOUCHDOWN = 1,   // Two touches are down
MTG_LIFTOFF = 2,     // Two touches lift off
MTG_SCROLLH  = 3,    // Horizontal scroll increment/decrement
MTG_SCROLLV = 4,     // Vertical scroll increment/decrement
MTG_PINCH = 5,       // Pinch/stretch increment/decrement
MTG_TAP= 6,          // Generated when a up follows a down = click
MTG_DOUBLETAP = 7,   // Two fingers tapped the screen twice
MTG_ROTATE = 8,      // Not implemented yet
```

Event Handler

```
OnGesture (int tGesture, int tAmountRelative, float tX, float tY, unsigned
int tElapsed_ms)
```

The application should contain an event handler with the above parameters and use SetGestureMode( ) (see below) to send a pointer to the event handler to NextWindow's DLL. The DLL will call the event handler when a gesture is detected and repeatedly as the gesture proceeds.

Parameters

`int tGesture`	`Type of gesture`
`int tAmountRelative`	`Amount the touch has moved/stretched since starting`
`float tX`	`Current X position of centroid of two touches`
`float tY`	`Current Y position of centroid of two touches`
`int tElapsed_ms`	`Milliseconds since gesture started`

Gesture Functions

The gesture functions allow applications to detect gesture events and to set the sensitivity of the gesture interpretation.

SetGestureMode ( )

Sets the gesture mode for receiving gesture events.

```
int  SetGestureMode (int tDeviceID, int tMode, ONGESTURE tCallbackPtr )
```

Returns SUCCESS or an error code (see Table 3 on page 9).

Parameters

`int tDeviceID`	`Serial number of the device`
`int tMode`	`Single-touch mode when no gesture received` `Either RM_MOUSE or RM_DITISER`

```
ONGESTURE tCallbackPtr   Pointer to the event handler function
```

Example

```
SetGestureMode(deviceID, RM_DIGITISER, &OnGestureEvent)
```

SetGestureThreshold ( )

Sets the threshold distance at which the gesture activates from the touch down position.

```
Void SetGestureThreshold (int GestureType, float tPercent )
```

Returns SUCCESS or an error code (see Table 3 on page 9).

Example

```
SetGestureThreshold (MTG_SCROLLH, 5)         //set hscroll to 5%
```

The hscroll gesture will start when the touches move 5% of the screen width from the touch-down position.

SetGestureGain ( )

This sets the gain of the gesture (smaller % results in faster acting)

```
Void SetGestureGain (int GestureType, float tPercentGain)
```

Returns SUCCESS or an error code (see Table 3 on page 9).

Example

```
SetGestureGain (MTG_SCROLLH, 5)        //set hscroll to 5% increments
```

Causes an hscroll event to occur every time two fingers move 5% of the screen width.

SetDisplayRect ( )

Sets the display region (in pixels) to be used for gestures.

```
Void SetDisplayRect (rect_t tDisplayRect)
```

Returns SUCCESS or an error code (see Table 3 on page 9).

Example

```
SetDisplayRect (DisplayRect)
```

Note: This function can be used to configure the touch region on a secondary monitor. e.g., 1024,0,2047,768

(See “Display Screen Coordinates” on page 8.)

Tutorial

The following example code illustrates the steps a typical application needs to take to receive touch information using the API calls in C++.

Step 1 – Include the Header file.

```
//-------------------------------------------------------------------------
#include “NWMultiTouch.h”
//-------------------------------------------------------------------------
```

Step 2 – Define the Event Handler to Receive Touch Information

```
//-------------------------------------------------------------------------
void __stdcall ReceiveMultiTouchData (DWORD deviceID, DWORD deviceStatus,
DWORD packetID, DWORD touches, DWORD ghostTouches)
{
  //Code to retrieve touch information will go here….
}
//-------------------------------------------------------------------------
```

Step 3 – Connect to the Touch Screen

```
//-------------------------------------------------------------------------
  DWORD result = 0;
  DWORD serialNum = 0;
    //Get the number of connected devices.
  DWORD numDevices = GetConnectedDeviceCount ();
    //If we have at least one connected device then try to connect to it.
  if (numDevices > 0)
  {
//Get the serial number of the device which uniquely identifies the device.
    serialNum = GetConnectedDeviceID (0);
//Initialise the device, passing in the serial number that uniquely
//identifies it and the event handler for processing touch packets.
    result = OpenDevice (serialNum, &ReceiveMultiTouchData);
  }
//-------------------------------------------------------------------------
```

Step 4 – Retrieve Touch Information

```
//-------------------------------------------------------------------------
void __stdcall ReceiveMultiTouchData (DWORD deviceID, DWORD deviceStatus,
DWORD packetID, DWORD touches, DWORD ghostTouches)
{
  if (deviceStatus == DS_TOUCH_INFO)
    {
    //Loop through all the touches.
    for (int tch = 0; touches > 0 && tch < MAX_TOUCHES; tch++)
    {
      //If the bit is set then a touch with this ID exists.
      if (touches & (1<<tch))
      {
        //Get the touch information.
        NWTouchPoint touchPt;
        DWORD retCode = GetTouch (deviceID, packetID,&touchPt,(1<<tch),0);
        if (retCode == SUCCESS)
        {
          //Do something with the touch information….
        }
      }
    }
    //Now loop through all the ghost touches.
    for (int tch = 0; ghostTouches > 0 && tch < MAX_TOUCHES; tch++)
    {
      //If the bit is set then a ghost touch with this ID exists.
      if (ghostTouches & (1<<tch))
      {
        NWTouchPoint ghostTouchPt;
        DWORD retCode = GetTouch (deviceID, packetID, &ghostTouchPt, 0,
                                  (1<<tch));
        if (retCode == SUCCESS)
        {
          //Do something with the ghost touch information.
        }
      }
    }

(Continued …)
```

```
    //Now, let's get the centroid of all these touches/ghost touches.
    NWTouchPoint centroidTouch;
    DWORD retCode = GetCentroid (deviceID, packetID, &centroidTouch,
                                 touches, ghostTouches);
    //If successful retrieval of the centroid data.
    if (retCode == SUCCESS)
    {
      //Do something with the centroid information
    }
   }
}
//---------------------------------------------------------------------------
```

Step 5 – Cleanup

```
void __fastcall TFrmMultiTouch::FormDestroy(TObject *Sender)
{
  // Close any open devices.
  for (int i = 0; i < MAX_DEVICES; i++)
  {
    if (connectedDevices[i] != NULL)
    {
      CloseDevice (connectedDevices[i]->DeviceID);
      delete connectedDevices[i];
      connectedDevices[i] = NULL;
    }
  }
  //Reset the device connect/disconnect event handlers.
  SetConnectEventHandler (NULL);
  SetDisconnectEventHandler (NULL);
}
```

Complete Code Example

Commented C++ and C# source code for an example application can be downloaded from:

http://www.nextwindow.com/support/application_notes/api.html

Future Developments

Possible developments that may be implemented in the future.

Retrieve Hardware Capabilities

Future versions of the firmware will be able to report the hardware capabilities of the device.

```
Bool GetHardwareCapabilities (Dword deviceID, NWHardwareSpecs*
hardwareSpecs);
```

The NWHardwareSpecs structure is shown below. More members may be added to this structure at a later date.

```
typedef
{
Dword  maxNumberTouches;
BooldigitizerAvailable;
Bool   highRes;
} NWHardwareSpecs;
```

Altering Number of Packets Stored

At present, a total of 255 packets are stored at any one time in the queue. Once this limit has been reached, the oldest packet is removed every time a new packet is processed.

In future versions, the application will be able to call the following function to set the number of packets that are stored at any one time.

```
Bool ChangeNumberOfPacketsStored (Dword numPackets);
```

A function to allow the application to retrieve the number of the packets stored will also be available in the following form.

```
Dword GetNumberOfPacketsStored ( );
```