

1 Data Enumerations and Structures

1.1 *WacomMTErrors*

Error conditions used as a return value for all functions.

```
typedef enum _WacomMTErrors
{
    WMTErrorsSuccess                = 0
    WMTErrorsDriverNotFound          = 1
    WMTErrorsBadVersion              = 2
    WMTErrorsAPIOutdated             = 3
    WMTErrorsInvalidParam            = 4
    WMTErrorsQuit                    = 5
    WMTErrorsBufferTooSmall          = 6
} WacomMTErrors
```

WMTErrorsSuccess:

Returned when a call is successful.

WMTErrorsDriverNotFound:

Returned when the API does not find an installed and running driver.

WMTErrorsBadVersion:

Returned when the version of the driver is not compatible with the API. This will happen when an application requests newer API data structures that are not supported by the driver.

WMTErrorsAPIOutdated:

Returned when an application requests API data structures that are no longer supported by the driver.

WMTErrorsInvalidParam:

Returned when one or more parameters are invalid.

WMTErrorsQuit:

Returned by wait functions when the API quit call is made or if the API has not been successfully initialized.

WMTErrorsBufferTooSmall

Returned if the supplied buffer is too small

1.2 *WacomMTDeviceType*

Type of sensor device. Used in the capabilities data structure.

```
typedef enum _WacomMTDeviceType
{
    WMTDeviceTypeOpaque              = 0
    WMTDeviceTypeIntegrated           = 1
} WacomMTDeviceType
```

WMTDeviceTypeOpaque:

The touch sensor is not integrated with a display. Opaque track pad like devices return this value.

WMTDeviceTypeIntegrated:

The touch sensor is integrated with a display. On screen touch devices, such as TabletPCs return this value.

1.3 *WacomMTCapabilityFlags*

Used in the capabilities structure to indicate the type of data supported by the device.

```
typedef enum _WacomMTCapabilityFlags
{
    WMTCapabilityFlagsRawAvailable          = (1 << 0)
    WMTCapabilityFlagsBlobAvailable         = (1 << 1)
    WMTCapabilityFlagsSensitivityAvailable  = (1 << 2)
    WMTCapabilityFlagsReserved              = (1 << 31)
} WacomMTCapabilityFlags
```

WMTCapabilityFlagsRawAvailable:

If this flag is set, the device supports raw data and raw data can be read.

WMTCapabilityFlagsBlobAvailable:

If this flag is set, the device supports blob data and blob data can be read.

WMTCapabilityFlagsSensitivityAvailable:

If this flag is set, sensitivity data will be available in the finger data. Sensitivity data is a value between 0 and 0xFFFF. If this flag is not set, sensitivity data will be set to zero in each up packet and max for the down/hold packets. See the WacomMTFinger data structure for a definition of sensitivity.

1.4 *WacomMTCapability*

This structure contains the physical and logical capabilities of a multi-touch device.

```
typedef struct WacomMTCapability
{
    int                Version
    int                DeviceID
    WacomMTDeviceType  Type
    float              LogicalOriginX
    float              LogicalOriginY
    float              LogicalWidth
    float              LogicalHeight
    float              PhysicalSizeX
    float              PhysicalSizeY
    int                ReportedSizeX
    int                ReportedSizeY
}
```

```
    int                ScanSizeX
    int                ScanSizeY
    int                FingerMax
    int                BlobMax
    int                BlobPointsMax
    WacomMTCapabilityFlags  CapabilityFlags
} WacomMTCapability
```

Version:

The version of this data structure. At this time this value should be 1.

DeviceID:

A value that identifies a touch device. This is a unique number that may vary from machine to machine and session to session, but will be the same during any given session. This value is used with all other calls to identify the device.

Type:

A value that indicates the device type. An opaque device does not have a fixed relationship with the screen. An integrated device is has a one to one relationship to a single monitor.

LogicalOriginX:

Minimum horizontal value of the device reported after interpolation. For an opaque device this value will be zero. For integrated devices this value is the (upper) left point in pixels of the mapped monitor in relation to the entire desktop. This may map to an adjacent monitor if the touch device is larger than the integrated display.

LogicalOriginY:

Minimum vertical value of the device reported after interpolation. For an opaque device this value will be zero. For integrated devices this value is the upper (left) point in pixels of the mapped monitor in relation to the entire desktop. This may map to an adjacent monitor if the touch device is larger than the integrated display.

LogicalWidth:

Width of the device reported after interpolation. For an opaque device this value is 1 and is unit neutral. For integrated devices this value is the number of pixels the device covers. This may map to an adjacent display if the touch device is larger than the integrated display.

LogicalHeight:

Height of the device reported after interpolation. For an opaque device this value is 1 and is unit neutral. For integrated devices this value is the number of pixels the device covers. This may map to an adjacent display if the touch device is larger than the integrated display.

PhysicalSizeX:

Width of the sensing area of the device in mm. Used with other size factors to convert the device data from one coordinate system to another.

PhysicalSizeY:

Height of the sensing area of the device in mm. Used with other size factors to convert the device data from one coordinate system to another.

ReportedSizeX:

Width of the device in native counts. Used with other size factors to determine the maximum resolution of the data. The horizontal resolution is calculated by dividing this value by PhysicalSizeX.

ReportedSizeY:

Height of the device in native counts. Used with other size factors to determine the maximum resolution of the data. The vertical resolution is calculated by dividing this value by PhysicalSizeY.

ScanSizeX:

Width of the device in scan coils. This is only provided for devices that support raw data. This value is used to calculate the size of the raw data buffer. (ScanSizeX * ScanSizeY) If the device does not support raw data this value should be ignored.

ScanSizeY:

Height of the device in scan coils. This is only provided for devices that support raw data. This value is used to calculate the size of the raw data buffer. (ScanSizeX * ScanSizeY) If the device does not support raw data this value should be ignored.

FingerMax:

Maximum number of fingers that are supported. This is the number of fingers this device can report as down at any given time. The can be used to help calculate the maximum size of the finger collection structure. This is the **not** the maximum value for FingerID.

BlobMax:

Maximum number of blobs in a blob aggregate. A blob is a series of points that define its outline. A blob can be a primary blob or a void blob. Each void blob has one and only one parent while a primary blob can have zero or more child blobs which define void areas within the parent blob. For example, a doughnut shape is a blob aggregate with the outer ring being the parent blob and the inner ring being a child blob. If the device does not support blob data this value should be ignored.

BlobPointsMax:

Maximum number of blob points that make up a blob. These blob values (BlobMax and BlobPointsMax) can be used to calculate the maximum size of a blob aggregate. (BlobMax * BlobPointsMax) If the device does not support blob data this value should be ignored.

CapabilityFlags:

A value to indicate the presence of specific data elements. See the WacomMTCapabilityFlags enumeration definition.

1.5 WacomMTFingerState

Used in the finger structure to indicate the state of the finger.

```
typedef enum _WacomMTFingerState
{
    WMTFingerStateNone      = 0
    WMTFingerStateDown      = 1
    WMTFingerStateHold      = 2
    WMTFingerStateUp        = 3
} WacomMTFingerState
```

WMTFingerStateNone:

A finger buffer may contain room for more than one contact. Any extra unused contact is set to

"None". Once a contact has passed through the "Up" state it has the TouchState set to "None" to indicate that no further processing is needed. Any other data included with this state is not valid.

WMTFingerStateDown:

Indicates initial finger contact. First touch packet for a particular contact.

WMTFingerStateHold:

Subsequent packets during the finger contact.

WMTFingerStateUp:

Last touch packet for a particular finger contact. Reported when the finger is lifted. This will be reported at the last known valid position.

1.6 WacomMTFinger

This structure contains the data for individual touch contacts.

```
typedef struct WacomMTFinger
{
    int                FingerID
    float              X
    float              Y
    float              Width
    float              Height
    unsigned short     Sensitivity
    float              Orientation
    bool               Confidence
    WacomMTFingerState TouchState
} WacomMTFinger
```

FingerID:

Unique identifier of the contact. This value starts with 1 for the first contact and increments for each subsequent contact. This value resets to 1 when all contacts are lifted up. This is to be used to track contacts from frame to frame. This does not represent a unique value for a specific finger (such as a ring finger) but is unique for the contact and represents the same finger for the duration of the contact.

X:

Scaled X of the contact area in logical units.

Y:

Scaled Y of the contact area in logical units.

Width:

Width of the contact area in logical units.

Height:

Height of the contact area in logical units.

Sensitivity:

Strength of the contact. This is **not** pressure. This is a device/user specific indication of the strength of the contact point. Only valid in relation to other fingers within the same

frame/gesture.

Orientation:

The orientation of the contact point in degrees.

Confidence:

If true the driver believes this is a valid touch from a finger. If false the driver thinks this may be an accidental touch, forearm or palm.

TouchState:

The state of this finger contact. See the WacomMTFingerState enumeration definition.

1.7 WacomMTFingerCollection

This structure allows for a list of fingers.

```
typedef struct WacomMTFingerCollection
{
    int             Version
    int             DeviceID
    int             FingerCount
    WacomMTFinger   *FingerData
} WacomMTFingerCollection
```

Version:

The version of this data structure. At this time this value should be 1.

DeviceID:

A value that identifies a touch device. This is a unique number that may vary from machine to machine and session to session, but will be the same during any given session.

FingerCount:

The number of elements in the finger data array. This value will vary from frame to frame but will never be greater than the FingerMax value for the specific device.

FingerData:

Pointer to an array of fingers. The size of the FingerData block is FingerCount * sizeof(WacomMTFinger).

1.8 WacomMTBlobType

Used by the blob structure to identify the blob type.

```
typedef enum _WacomMTBlobType
{
    WMTBlobTypePrimary      = 0
    WMTBlobTypeVoid         = 1
} WacomMTBlobType
```

WMTBlobTypePrimary:

This is an outline of an outer primary blob.

WMTBlobTypeVoid:

This is an outline of an inner blob that is contained within an outer primary blob. Void blobs are regions of no touch within primary blobs. A primary blob may contain one or more void blobs. Void blobs do not contain other blobs. A unique primary blob can be within another blobs void but the void does not reference that blob.

1.9 *WacomMTBlobPoint*

This structure contains information about a specific point of blob data.

```
typedef struct WacomMTBlobPoint
{
    float          X
    float          Y
    unsigned short  Sensitivity
} WacomMTBlobPoint
```

X:

Scaled X value of a blob point in logical units.

Y:

Scaled Y value of a blob point in logical units.

Sensitivity:

Strength of the signal at this blob point. This is **not** pressure. This is a device/user specific indication of the strength of the contact point. Only valid in relation to other blobs within the same frame.

1.10 *WacomMTBlob*

This structure contains the contact data for an irregular region.

```
typedef struct WacomMTBlob
{
    int          BlobID
    float        X
    float        Y
    bool         Confidence
    WacomMTBlobType BlobType
    int          ParentID
    int          PointCount
    WacomMTBlobPoint *BlobPoints
} WacomMTBlob
```

BlobID:

This is a value that uniquely identifies this blob. This value persists from frame to frame.

X:

Scaled X center of gravity of the blob area in logical units.

Y:

Scaled Y center of gravity of the blob area in logical units.

Confidence:

If true the driver believes this is a valid touch. If false the driver thinks this may be an accidental touch, forearm or palm.

BlobType:

The blob type this structure represents. See the WacomMTBlobType enumeration definition.

ParentID:

This identifies the parent blob. Valid only if the BlobType is "Void".

PointCount:

The number of elements in the blob points array. This is the number of points that make up the outline of the blob.

BlobPoints:

Pointer to an array of blob points. The size of the BlobPoints block is PointCount * sizeof(WacomMTBlobPoint). The blob points form a closed area.

1.11 WacomMTBlobAggregate

This structure allows for a list of blobs.

```
typedef struct WacomMTBlobAggregate
{
    int                Version
    int                DeviceID
    int                BlobCount
    WacomMTBlob        *BlobData
} WacomMTBlobAggregate
```

Version:

The version of this data structure. At this time this value should be 1.

DeviceID:

A value that identifies a touch device. This is a unique number that may vary from machine to machine and session to session, but will be the same during any given session.

BlobCount:

Number of elements in the blob data array.

BlobData:

An array of blobs. The size of the BlobData block is BlobCount * sizeof(WacomMTBlob).

1.12 WacomMTRawData

This structure represents the strength values for the surface of the device.

```
typedef struct WacomMTRawData
{
    int                Version
    int                DeviceID
    int                ElementCount
```



```

    unsigned short    *Sensitivity
} WacomMTRawData

```

Version:

The version of this data structure. At this time this value should be 1.

DeviceID:

A value that identifies a touch device. This is a unique number that may vary from machine to machine but will be the same during any given session.

ElementCount:

Number of elements in the sensitivity array. This value should be ScanSizeY * ScanSizeX.

Sensitivity:

Pointer to an array sensitivity values. The size of the Sensitivity block is ElementCount * sizeof(unsigned short). The location of each point is calculated as (Y * ScanSizeX) + X.

1.13 WacomMTHitRect

This structure represents a hit test rectangle.

```

typedef struct WacomMTHitRect
{
    float    left
    float    top
    float    right
    float    bottom
} WacomMTHitRect

```

left:

The left bound of the rectangle.

top:

The top bound of the rectangle.

right:

The right bound of the rectangle.

bottom:

The bottom bound of the rectangle.

1.14 WacomMTProcessingMode

Used by the callback functions. Provides instructions for how to process the data. If no flags are set the data is sent to the callback and not processed by the system.

```

typedef enum _WacomMTProcessingMode
{
    WMTProcessingModeNone           = 0
    WMTProcessingModeObserver       = (1 << 0)
    WMTProcessingModeReserved       = (1 << 31)
} WacomMTProcessingMode

```

WMTProcessingModeNone:

The data is sent to the callback and no other processing is done.

WMTProcessingModeObserver:

The data is posted in parallel to the OS for processing.

2 Functions

2.1 *WacomMTInitialize*

WacomMTErr WacomMTInitialize(int libraryAPIVersion);

This function attempts to connect the application to the driver and must be called successfully before any other Wacom Multi-Touch function is called.

Parameters:

libraryAPIVersion:

This is the version of the API that the application is using. This value will be used by the API to construct the expected data structures for the application. Please use the provided predefined value WACOM_MULTI_TOUCH_API_VERSION.

Return Value:

See the WacomMTErr enumeration definition.

2.2 *WacomMTQuit*

void WacomMTQuit();

This function closes the connection to the driver service. This should be called when the application is closing or no longer needs touch data. After calling WacomMTQuit, WacomMTInitialize would need to be called again to resume touch interaction.

2.3 *WacomMTGetAttachedDeviceIDs*

int WacomMTGetAttachedDeviceIDs(int *deviceArray, size_t bufferSize);

This function returns the number of multi-touch sensors attached to the system.

Parameters:

deviceArray:

A user allocated buffer used to return the deviceIDs of the currently attached devices. This call will provide as many devices as will fit into the provided buffer. This can be NULL.

bufferSize:

The size of the buffer provided. Should be zero if no buffer provided.

Return Value:

The return value is the number of sensors attached to the system.

2.4 *WacomMTGetDeviceCapabilities*

WacomMTErr WacomMTGetDeviceCapabilities(int deviceID, WacomMTCapability *capabilityBuffer);

This function fills in a caller allocated WacomMTCapability structure with the capability information for the requested device identifier.

Parameters:

deviceID:

The ID of the device. These IDs can be from the GetAttachedDeviceIDs array, the finger packet deviceID member, the blob packet deviceID member or the raw packet

deviceID member.

capabilityBuffer:

This is the caller allocated structure that is filled in with device capability data upon success. This structure will be defined by the API version provided at initialization.

Return Value:

See the WacomMTErrors enumeration definition.

2.5 WacomMTRegisterAttachCallback

WacomMTErrors WacomMTRegisterAttachCallback(WMT_ATTACH_CALLBACK attachCallback, void *userData);

This function allows you to register a callback function that you would like called when a new touch device is attached. When registered the API will issue a callback for each device currently attached. Only one attach callback can be registered for each process. To cancel the attach callback, a process can call this function with NULL.

Parameters:

The function definition is: void (*WMT_ATTACH_CALLBACK)(WacomMTCapability deviceInformation, void *userData);

This function will be called with the WacomMTCapability structure for the device attached and the userData provided when the call was registered.

Return Value:

See the WacomMTErrors enumeration definition.

2.6 WacomMTRegisterDetachCallback

WacomMTErrors WacomMTRegisterDetachCallback(WMT_DETACH_CALLBACK detachCallback, void *userData);

This function allows you to register a callback function that you would like called when a touch device is detached. Only one detach callback can be registered for each process. To cancel the detach callback, a process can call this function with NULL.

Parameters:

The function definition is: void (*WMT_DETACH_CALLBACK)(int deviceId, void *userData);

The function will be called with the deviceID of the sensor that has been detached and the userData provided when the call was registered.

Return Value:

See the WacomMTErrors enumeration definition.

2.7 WacomMTRegisterFingerReadCallback

WacomMTErrors WacomMTRegisterFingerReadCallback(int deviceID, WacomMTHitRect *hitRect, WacomMTProcessingMode mode, WMT_FINGER_CALLBACK fingerCallback, void *userData);

This function allows you to register a callback function that you would like called when a finger touch packet is ready and within the requested device's hit rectangle. A process can create as many callbacks as needed but only one callback per device hit rectangle. If you wish to cancel a callback, call this function on an existing device hit rectangle with NULL for the function.

The callbacks are processed in the order in which they are created.

Parameters:

deviceID:

The deviceID of the device.

hitRect:

A rectangle used to hit test the data. If a finger begins contact with in the hit rectangle the callback will be called. The callback will continue to be called for that finger until that contact is removed from the device. If a contact begins outside the hit rectangle, the callback will not be called for that finger. A NULL rectangle will assume the entire device surface. The hit rectangle is defined in the logical units of the device. See WacomMTCapability structure for a definition of logical units.

mode:

Specifies what the API does will the data during the callback.

fingerCallback:

The function definition is: `int (*WMT_FINGER_CALLBACK)(WacomMTFingerCollection *fingerPacket, void *userData);`

The function will be called with the fingerPacket structure filled in and the userData provided when the call was registered. The fingerPacket is only valid while the callback is being processed. This memory is release/reused upon return from the callback. The return value is reserved and should be zero.

userData:

A parameter provided by the caller and echoed back in the callback function.

Return Value:

See the WacomMTError enumeration definition.

2.8 WacomMTRegisterBlobReadCallback

WacomMTError WacomMTRegisterBlobReadCallback(int deviceID, WacomMTHitRect *hitRect, WacomMTProcessingMode mode, WMT_BLOB_CALLBACK blobCallback, void *userData);

This function allows you to register a callback function that you would like called when blob data is ready and within the requested device's hit rectangle. A process can create as many callbacks as needed but only one callback per device hit rectangle. If you wish to cancel a callback, call this function on an existing device hit rectangle with NULL for the function. The callbacks are processed in the order in which they are created.

Parameters:

deviceID:

The deviceID of the device.

hitRect:

A rectangle used to hit test the data. If any part of the blob contact is within the hit rectangle the callback will be called. A NULL rectangle will assume the entire device surface. The hit rectangle is defined in the logical units of the device. See WacomMTCapability structure for a definition of logical units.

mode:

Specifies what the API does will the data during the callback.

blobCallback:

The function definition is: `int (*WMT_BLOB_CALLBACK)(WacomMTBlobAggregate *blobPacket, void *userData);`

The function will be called with the blobPacket structure filled in and the userData provided when the call was registered. The blobPacket is only valid while the callback is being processed. This memory is release/reused upon return from the callback. The return value is reserved and should be zero.

userData:

A parameter provided by the caller and echoed back in the callback function.

Return Value:

See the WacomMTError enumeration definition.

2.9 WacomMTRegisterRawReadCallback

WacomMTError WacomMTRegisterRawReadCallback(int deviceID, WacomMTProcessingMode mode, WMT_RAW_CALLBACK rawCallback, void *userData);

This function allows you to register a callback function that you would like called when raw data is ready for the specified device. Only one callback can be registered per device. If you wish to cancel the device callback call this function with NULL for the callback. The callbacks are processed in the order in which they are created.

Parameters:*deviceID:*

The deviceID of the device.

mode:

Specifies what the API does with the data during the callback.

rawCallback:

The function definition is: `int (*WMT_RAW_CALLBACK)(WacomMTRawData *rawPacket, void *userData);`

The function will be called with the rawPacket structure filled in and the userData provided when the call was registered. The rawPacket is only valid while the callback is being processed. This memory is release/reused upon return from the callback. The return value is reserved and should be zero.

userData:

A parameter provided by the caller and echoed back in the callback function.

Return Value:

See the WacomMTError enumeration definition.

2.10 WacomMTRegisterFingerReadHWND (This is a Windows only call)

WacomMTError WacomMTRegisterFingerReadHWND(int deviceID, WacomMTProcessingMode mode, HWND hWnd, int bufferDepth);

This function allows you to register a window handle to receive finger data. When a packet of finger data for the given device is ready a WM_FINGERDATA message will be sent to the window handle. The wParam parameter will be a pointer to a WacomMTFingerCollection structure that contains the finger data. The bufferDepth parameter determines the number of

callback buffers created. The buffers are used in a ring format. Only one finger data callback is allowed for each device per window handle. The window handle is also used to produce a hit rectangle. The hit rectangle is calculated when this function is called. If the window is moved or resized this function should be called again. If the `bufferDepth` is changed, all existing buffers will be invalid. To cancel the device callback this function should be called with the `bufferDepth` of zero.

Parameters:

deviceID:

The `deviceID` of the device.

mode:

Specifies what the API does with the data during the callback.

hWnd:

This is the window handle that will receive the `WM_FINGERDATA` message.

bufferDepth:

This is the number of `WacomMTFingerCollection` data structures that will be allocated for the message callbacks.

Return Value:

See the `WacomMTError` enumeration definition.

2.11 **WacomMTRegisterBlobReadHWND** (This is a Windows only call)

**WacomMTError WacomMTRegisterBlobReadHWND(int deviceID,
WacomMTProcessingMode mode, HWND hWnd, int bufferDepth);**

This function allows you to register a window handle to receive blob data. When a blob packet for the given device is ready a `WM_BLOBDATA` message will be sent to the window handle. The `wParam` parameter will be a pointer to a `WacomMTBlobAggregate` structure that contains the blob data. The `bufferDepth` parameter determines the number of callback buffers created. The buffers are used in a ring format. Only one blob data callback is allowed for each device per window handle. The window handle is also used to produce a hit rectangle. The hit rectangle is calculated when this function is called. If the window is moved or resized this function should be called again. If the `bufferDepth` is changed, all existing buffers will be invalid. To cancel the device callback this function should be called with the `bufferDepth` of zero.

Parameters:

deviceID:

The `deviceID` of the device.

mode:

Specifies what the API does with the data during the callback.

hWnd:

This is the window handle that will receive the `WM_BLOBDATA` message.

bufferDepth:

This is the number of `WacomMTBlobAggregate` data structures that will be allocated for the message callbacks.

Return Value:

See the `WacomMTError` enumeration definition.

2.12 WacomMTRegisterRawReadHWND (This is a Windows only call)

**WacomMTError WacomMTRegisterRawReadHWND(int deviceID,
WacomMTProcessingMode mode, HWND hWnd, int bufferDepth);**

This function allows you to register a window handle to receive raw data. When a data frame for the given device is ready a WM_RAWDATA message will be sent to the window handle. The wParam parameter will be a pointer to a WacomMTRawData structure. The bufferDepth parameter determines the number of callback buffers created. The buffers are used in a ring format. Only one raw data callback is allowed for each device per window handle. If the bufferDepth is changed, all existing buffers will be invalid. To cancel the device callback this function should be called with the bufferDepth equal to zero.

Parameters:

deviceID:

The deviceID of the device.

mode:

Specifies what the API does with the data during the callback.

hWnd:

This is the window handle that will receive the WM_RAWDATA message.

bufferDepth:

This is the number of RawData data structures that will be allocated for the message callbacks.

Return Value:

See the WacomMTError enumeration definition.