

# 3DxWareMac SDK

## v1.0.4

January 2018

## Contents

0. Overview .....	3
1. Getting Started .....	3
1.1 3D Application Operating Modes .....	3
1.2 Alternate Orientations .....	4
2. Data Supplied .....	5
2.1 How Data Should be Used .....	5
2.2 Data Tuning .....	8
3. Compatibility with OS X HID Manager .....	9
4. Using the 3DxWareMac SDK .....	9
4.1 Step 1. Add 3DconnexionClient.framework to your project .....	9
4.2 Step 2. Open a connection to the driver .....	9
4.3 Step 3. Process events sent to your callback .....	11
4.4 Step 4. Close the driver connection .....	11
4.5 Weak linking 3DconnexionClient.framework in Xcode .....	12
4.6 Sandboxed applications .....	13
5. Functions .....	15
5.1 SetConnexionHandlers .....	15
5.2 CleanupConnexionHandlers .....	16
5.3 RegisterConnexionClient .....	16
5.4 SetConnexionClientMask .....	17
5.5 UnregisterConnexionClient .....	18
6. Callbacks .....	19
6.1 ConnexionAddedHandlerProc .....	19
6.2 ConnexionRemovedHandlerProc .....	19
6.3 ConnexionMessageHandlerProc .....	20
7. Data Types .....	21
7.1 ConnexionDeviceState .....	21
8. Constants .....	23
8.1 Device command values .....	23
9. Samples .....	24
9.1 3DxCubeDemo .....	24
9.2 3DxSNAxisDemo .....	24
9.3 3DxMultithreadedValues .....	24
9.4 Connexion Client Test .....	24
9.5 3DxValuesCarbon .....	24

## 0. Overview

The 3DxWareMac SDK provides the programming resources developers need to add 3DxWareMac navigation support for their applications.

With the 3DxWareMac SDK, you get:

- A 3DconnexionClient.framework framework API that supports Intel based Macs. The framework can be found in `/Library/Frameworks`. It is **installed with** the driver software **3DxWareMac!**
- This documentation that explains the API and how to use it inside your application.
- Several programming examples to get you started.

## 1. Getting Started

3Dconnexion navigation products are true 3D input devices that detect the slightest fingertip pressure and resolve the pressure into X, Y, and Z translations and rotations, moving your 3D models instantaneously and simultaneously. This provides intuitive, interactive six-degrees-of-freedom control of 3D graphical images and objects.

### 1.1 3D Application Operating Modes

Using a 3Dconnexion device is natural and intuitive. There are two basic ways to move objects in 3D with the device:

- Manipulate the 3Dconnexion controller cap as if you are holding the 3D model in your hand; this is **Object Mode**. Push left and the model moves left. Push right and it moves right. Lift up or push down and the model moves accordingly. Push away and or pull forward on the sensor and the model responds accordingly. Twist in any direction and the model rotates in that direction. This is the most natural mode when there is a single item to move.
- Manipulate the 3Dconnexion controller cap as if it is a camera or your head; this is **Camera Mode**. Push into the scene and the **camera** moves forward into the scene. The scene will appear to move toward and around the viewer. **Push** left and the **camera** moves to the left (the **scene** moves to the right). Push right and the **camera** moves to the right (the **scene** moves to the left). Lift up and the **camera** moves up. Push down and the **camera** moves down. The scene always moves the opposite direction of the input device. The viewer is entering the scene as if walking around in it. It normally takes some time to get used to this mode.

It is a natural mode in a virtual scene environment when there is a clear floor and/or horizon rather than some single object to move.

You can apply any of these actions at the same time, causing the image to move as you move.

## **1.2 Alternate Orientations**

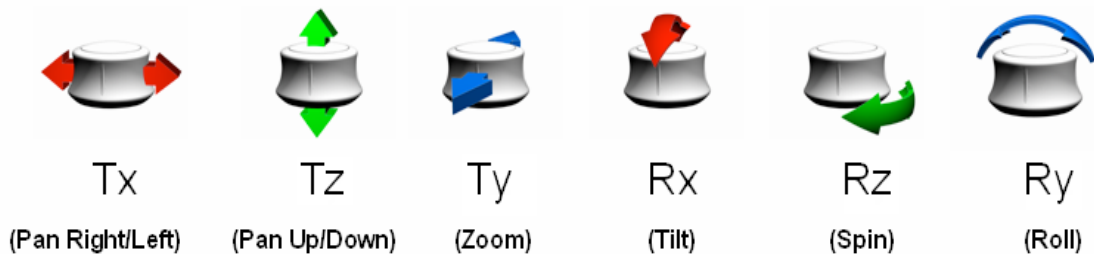
Some users prefer to think of the Finder desktop as being a real desktop. They like to think of their screen as looking down on their desktop, not as looking at a projection of it or say a whiteboard. Therefore, they prefer to have the controller cap oriented similar to a mouse in which pushing away from them causes the object under control to go *up* the screen in the same way that pushing a mouse away from them causes the mouse pointer to move up the screen. The 3DxWare driver GUI supports this option by rearranging the axes. Several axes have to be changed in conjunction to maintain a consistent interaction model.

You should develop your application using the orientation that puts Zoom towards you and away from you parallel to your desk. The GUI will then rearrange the axes for your end-users.

## 2. Data Supplied

3Dconnexion devices provide full, simultaneous six-axis movement in any and all directions. The diagrams below show the orientation of Translation and Rotation axes on the sensor. Following the diagrams, there are two charts explaining the data range of each axis, and the axis 'meaning' in both 6D Navigation (Camera Mode) and 6D Object Control (Object Mode).

The driver provides data with 10 bits of resolution, therefore allowing for high precision movements. It is crucial to a good implementation of the 3Dconnexion device in your application that the movement resulting from pressures on the sensor be as smooth and instantaneous as possible. Please use the diagrams and charts below when adding support for the 3Dconnexion device to your application. The motions in parentheses are what are designated in the 3Dconnexion Control Panel by default and reflects the user terminology for these motions.



The normal range of the device axes is approximately +/- 500. However, the user can scale up or down these values in the GUI, so your application should be able to handle larger or smaller values.

### 2.1 How Data Should be Used

One of the most frequently asked questions 3Dconnexion receives is how to map the axes of the device to match the application. Here is the rule of thumb.

**Note:**









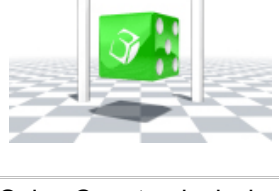



The home position for the device is with the negative Z-axis pointing towards the screen. The dice in these illustrations, when at home or rest, is between the two posts, slightly above the surface as shown in the graphic below.









**Object at Rest**

<b>Translation Controls</b>		
<b>Device Value</b>	<b>Camera Mode</b>	<b>Object Mode</b>
Translate Z-axis 0 to -MAX 	Camera Moves Backward 	Object Moves Closer 
Translate Z-axis 0 to MAX 	Camera Moves Forward 	Object Moves Away 
Translate Y-axis 0 to -MAX 	Camera Moves Up, Jumps or Flies 	Object Moves Up 
Translate Y-axis 0 to MAX 	Camera Moves Down or Crouches 	Object Moves Down 
Translate X-axis 0 to MAX 	Camera Moves Right 	Object Moves Right 
Translate X-axis 0 to -MAX 	Camera Moves Left 	Object Moves Left 



Rotation Control		
Device Value	Camera Mode	Object Mode
Rotate Z-axis 0 to -MAX 	Left Cartwheel or Barrel Roll 	Spins Counterclockwise 
Rotate Z-axis 0 to MAX 	Right Cartwheel or Barrel Roll 	Spins Clockwise 
Rotate Y-axis 0 to MAX 	Spin Clockwise 	Spins Clockwise 
Rotate Y-axis 0 to -MAX 	Spin Counterclockwise 	Spins Counterclockwise 

<p>Rotate X-axis 0 to MAX</p> 	<p>Pitch Up or Look Up</p> 	<p>Object top spins towards you</p> 
<p>Rotate X-axis 0 to -MAX</p> 	<p>Pitch Down or Look Down</p> 	<p>Object bottom spins towards you</p> 

## 2.2 Data Tuning

It is important, from the point-of-view of user experience, that the speed response of one application is consistent with that of the others. A user easily relates to the force necessary to achieve a given speed (rotation or translation) in an application and expects the same behavior from other applications.

### Note:

The initial slow default speed of the "Any Application" configuration is for the inexperienced user. What a developer may feel as a more comfortable speed may, in fact, be more difficult for an inexperienced user to control.

To tune the speed of an application, the developer should use the "Cube Demo" program as a reference.

A proven method is having the driver (3DxWareMac) using the "Any Application" configuration and setting the 'Overall Speed' in the driver to a value that the developer feels gives a good response in the Demo. Then, using the same configuration, the developer can compare the response of his application with that of the demo.



## 3. Compatibility with OS X HID Manager

The 3Dconnexion driver (3DxWareMac) and 3DxWareMac SDK are fully compatible with OS X's HID Manager technology. Applications can use either the HID manager or 3DxWareMac SDK to get device data.

## 4. Using the 3DxWareMac SDK

The 3DxWareMac framework is installed as part of the 3DxWareMac driver software. You can find it in  
`/Library/Frameworks/3DconnexionClient.framework.`

3DxWareMac SDK was designed to simplify the process of integrating a 3D navigation device with your application. The following steps will guide you through the integration process.

### ***4.1 Step 1. Add 3DconnexionClient.framework to your project***

Regardless of the development tool you're using, you will need to reference the 3DconnexionClient.framework. Please refer to your tools documentation on how to add a framework.

### ***4.2 Step 2. Open a connection to the driver***

The framework uses a callback mechanism to send your application button or axis events from the device. In order to receive these events, you must first register your callback function using `SetConnexionHandlers()`.

#### **Note:**

In previous framework versions this was done via `InstallConnexionHandler()`. This call will still work, but it is deprecated! Do not use it in new or updated applications!

```
OSErr result = SetConnexionHandlers(MyMessageHandler, MyAddedMessageHandler, MyRemovedMessageHandler, kUseSeparateThread);
```

- `MyMessageHandler` is your callback function that would receive the events.
- `MyAddedMessageHandler` is called whenever a device is added.
- `MyRemovedMessageHandler` is called whenever a device is added.

- *kUseSeparateThread* is a BOOL parameter supply true/TRUE/YES if you want to have MyMessageHandler process the 3D Mouse events on a separate thread, otherwise false/FALSE/NO.

Note:

*kUseSeparateThread* is new with 3DxWareMac 10.2.2.

Having the 3D Mouse events processed on a separate thread will eliminate motion events being “buffered”, which could have happened if the application had some serious big work to do on the very same thread. This is the case if the developer did not want to or had no possibility separate this work into different threads.

**IMPORTANT:**

If you choose to have the events processed on a separate thread, make sure that you take care about thread safety when exchanging data between the callbacks and your main thread. Also if you issue calls from within the callback into your main thread / other threads!

Next, register your application with the driver to start the flow of events.

```
UInt16 clientID = RegisterConnexionClient('MCTt', "\p3DxClientTest",
kConnexionClientModeTakeOver, kConnexionMaskAll);
```

RegisterConnexionClient() returns a unique ID to identify your application to the driver. You will need to save this ID and pass it back to driver when calling certain SDK functions.

You would also use this ID to filter events coming in. The driver will broadcast device events to all registered applications. Each event will have a *client* field that identifies the intended target for the event. The value of *client* is normally the ID of the front-most active application.

RegisterConnexionClient() takes four parameters.

From the example:

*'MCTt'* - Is the application's CFBundleSignature code.

*"\p3DxClientTest"* – Application's executable name.

*kConnexionClientModeTakeOver* - Reserved. Must be this constant.

*kConnexionMaskAll* – Tells the driver what type of events you're interested in receiving. Using *kConnexionMaskAll* means give me button and all axis events.

### 4.3 Step 3. Process events sent to your callback

```
void MyMessageHandler(io_connect_t connection, natural_t messageType, void
*messageArgument)
{
    ConnexionDeviceState *state;

    switch (messageType)
    {
        case kConnexionMsgDeviceState:
            state = (ConnexionDeviceState*)messageArgument;
            if (state->client == clientID)
            {
                // decipher what command/event is being reported by the driver
                switch (state->command)
                {
                    case kConnexionCmdHandleAxis:
                        // state->axis will contain values for the 6 axis
                        break;

                    case kConnexionCmdHandleButtons:
                        // state->buttons reports the buttons that are pressed
                        break;
                }
            }
            break;

        default:
            // other messageTypes can happen and should be ignored
            break;
    }
}
```

### 4.4 Step 4. Close the driver connection

When you no longer need the 3Dconnexion device, un-register your application and remove the callback handlers.

```
void UnregisterConnexionClient(clientID);
void CleanupConnexionHandlers();
```

#### **IMPORTANT:**

*All calls to the 3Dconnexion API **MUST** happen on the same thread. This thread **MUST** “live” as long as your application does or at least until you actively close the connection to the driver as described in step 4!*

## 4.5 Weak linking 3DconnexionClient.framework in Xcode

Most applications typically bind with a framework during load time. If your customer does not have the 3Dconnexion driver installed on his system, this will result in missing symbols for 3Dconnexion functions and your application will fail to launch.

To avoid this issue, you should weak link the 3DconnexionClient.framework and check for its presence at runtime. First define the one of the framework functions as weak-linked.

```
extern int16_t SetConnexionHandlers(  
    ConnexionMessageHandlerProc messageHandler,  
    ConnexionAddedHandlerProc addedHandler,  
    ConnexionRemovedHandlerProc removedHandler,  
    bool useSeparateThread) __attribute__((weak_import));
```

Then, before calling a framework function, check for its availability:

```
// Make sure the framework is installed  
if(SetConnexionHandlers != NULL)  
{  
    // Install message handler and register our client  
    error = SetConnexionHandlers(TestMessageHandler,  
                                TestDeviceAddedHandler,  
                                TestDeviceRemovedHandler,  
                                TEST_WITH_SEPARATE_THREAD);  
    ...  
}
```

Finally, be sure that the 3DconnexionClient.framework is not strongly linked in your project. To weak link the framework, follow the steps on this links: at Apple's developer web site:

<https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPFrameworks/Frameworks/Frameworks.html>

<https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WeakLinking.html>

### **IMPORTANT:**

**DO NOT INCLUDE** the 3DconnexionClient.framework in your application's bundle! This will break your application's functionality if the framework shipped with the driver changes!

**ALWAYS** weak link against and check for the availability of the framework!

## 4.6 Sandboxed applications

If your application is going to be distributed or sold via Apple's AppStore it is a requirement, that your applications is code signed as well as sandboxed.

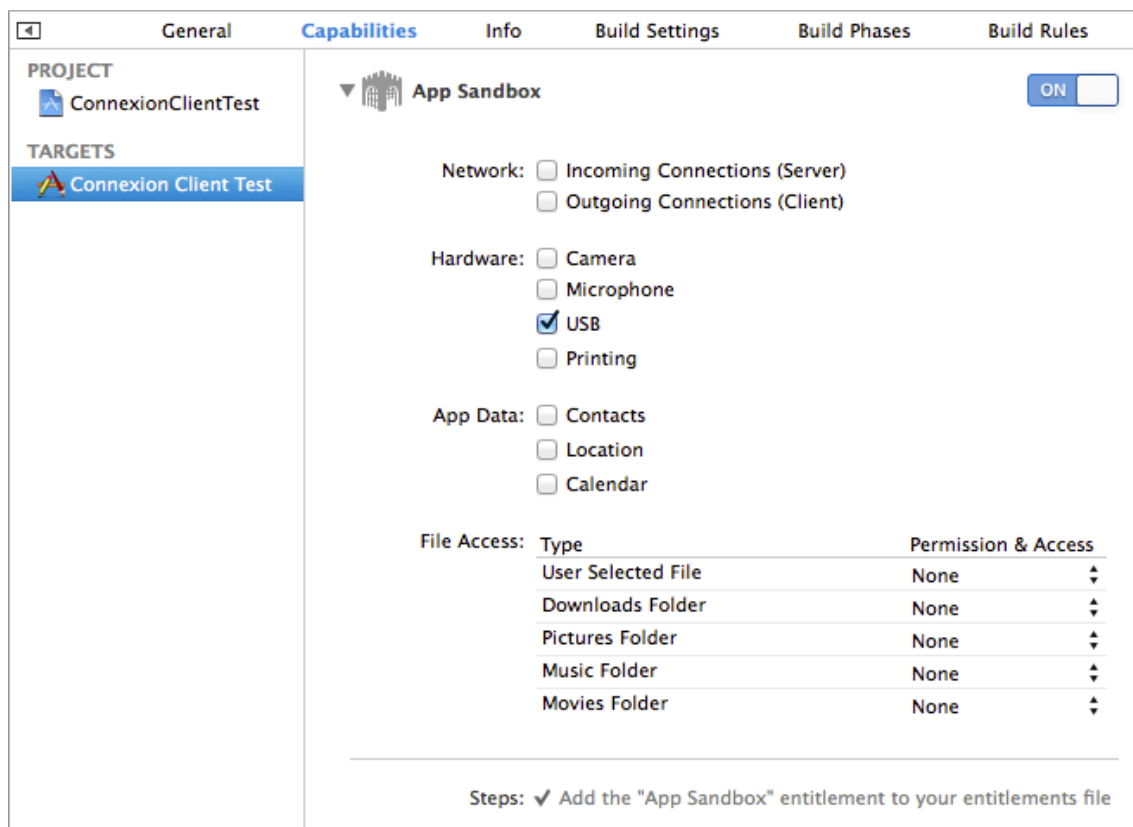
Having a sandboxed application is nice in terms of user safety/security, but also has some disadvantages form the developer's perspective:

The application is no longer allowed to access certain resources of your Mac; in case of 3DxWareMac / SDK the sample projects we supplied with this SDK would no longer receive any events from a 3D mouse.

Note:

Since the sample projects are not sandboxed not code signed this is not an issue, but you can play with them and make them build sandboxed applications.

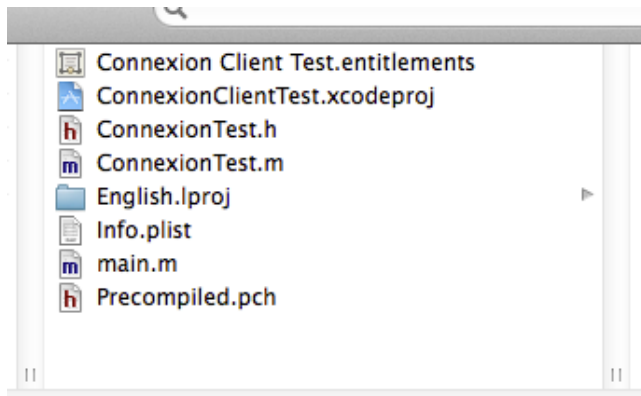
To again get 3D mouse events, you need to add an entitlement resource to your project, which allows “USB” hardware to be accessed again.



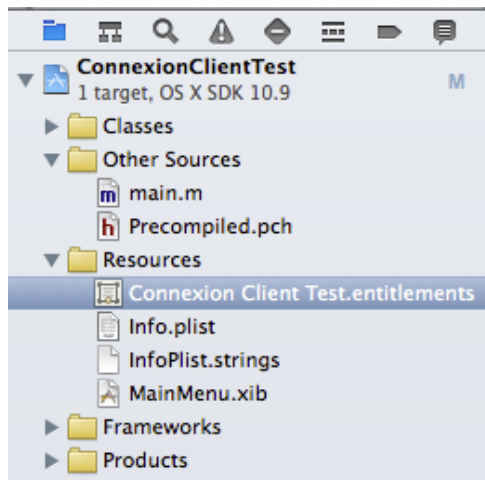
In Xcode this can easily be achieved in your application's target.

Under Capabilities (see image above). You'll need to set “App Sandbox” to “ON”. Now you can enable “USB” (category “Hardware”).

Xcode will create an entitlements file (a plist). It needs to be added to your Xcode project. For example “Connexion Client Test.entitlements”



Drag (or add via Xcode's menu “Add Files to ...”) this file to your Xcode project, for example to your “Resource” group/folder.



You're (almost) done.

To make sandboxing and the USB entitlement effective, you'll also need to code sign your application. To do so, you'll need to sign up for an Apple developer account and get the your developer certificate to sign your application.

Read more on App Sandbox in Apple's develop documentation:

<https://developer.apple.com/library/mac/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html>

<https://developer.apple.com/library/mac/documentation/Security/Conceptual/AppSandboxDesignGuide/AppSandboxQuickStart/AppSandboxQuickStart.html>

About Code signing:

<https://developer.apple.com/library/mac/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html>

## 5. Functions

### 5.1 SetConnexionHandlers

Registers your callback functions and lets you select if the events get processed in a separate thread.

```
int16_t SetConnexionHandlers(  
    ConnexionMessageHandlerProc messageHandler,  
    ConnexionAddedHandlerProc addedHandler,  
    ConnexionRemovedHandlerProc removedHandler,  
    bool useSeparateThread) __attribute__((weak_import));
```

#### Parameters

##### *messageHandler*

A pointer to your message handler callback function. This function will be called whenever the device cap is actuated or a device button is pressed or released

##### *addedHandler*

A pointer to your device was added callback function. This function is invoked when a 3Dconnexion device is plugged into the computer. If you are not interested in receiving this notification, pass NULL.

##### *removedHandler*

A pointer to your device was removed callback function. This function is invoked when a 3Dconnexion device is unplugged from the computer. If you are not interested in receiving this notification, pass NULL.

##### *useSeparateThread*

A bool value. If set to true, all 3D Mouse events are processed in a separate thread. In that case make sure that you use thread-safe calls within the callback, when exchanging data between or calling into other threads!

#### Return Value

A return value of 0 indicates success. Any other value means the call failed and your callback functions were not registered.

#### Discussion

The driver will notify your application when a 3Dconnexion device is installed or removed from the computer or when a device event (cap motion or button press / release) has occurred. Notification is done through callback functions that you register with the driver.

#### Declared In

ConnexionClientAPI.h

## 5.2 CleanupConnexionHandlers

Unregisters your callback functions

```
void CleanupConnexionHandlers(void);
```

### Parameters

None.

### Return Value

None.

### Discussion

Removes any callback functions that were registered using *InstallConnexionHandlers()*. You must call this function prior to exiting your application.

### Declared In

ConnexionClientAPI.h

## 5.3 RegisterConnexionClient

Registers your application with the driver

```
uint16_t RegisterConnexionClient(  
    uint32_t signature,  
    uint8_t *name,  
    uint16_t mode,  
    uint32_t mask  
);
```

### Parameters

*signature*

Your application's creator signature. This is the 32-bit CFBundleSignature value. You can pass zero (0) if your application does not have one. If you pass 0, you must provide a valid *name* parameter. The driver uses this value to identify if your application is the active (front-most) application.

*name*

This is your application's executable name formatted as a Pascal string (e.g. "\pMyApplication"). Note that this string is the actual executable and not the application bundle name. You may pass NULL but if you do, the signature parameter must be a valid CFBundleSignature value.

*mode*



Reserved parameter. You must pass the constant  
*kConnexionClientModeTakeOver*

*mask*

Specify what device events, buttons or axis motion, are to be sent to your application. This is a 32-bit value that can be OR'd together. See *ConnexionClient.h* for a list of the capability mask constants.

### Return Value

Upon success this

### Discussion

This function signals the driver to start sending device messages to your application. The callback functions you registered with *InstallConnexionHandlers* will be invoked appropriately.

### Declared In

*ConnexionClientAPI.h*

## 5.4 SetConnexionClientMask

Sets the capability mask bits your application will handle

```
void SetConnexionClientMask(  
    uint16_t clientID,  
    uint32_t mask  
);
```

### Parameters

*clientID*

ID returned from a call to *RegisterConnexionClient*.

*mask*

Specify what device events, buttons or axis motion, are to be sent to your application. This is a 32-bit value that can be OR'd together. See *ConnexionClient.h* for a list of the capability mask constants.

### Discussion

You can change the capabilities mask with this function. For example, you can specify that only translation data (X, Y, Z) should be sent to your application by passing *kConnexionMaskAxisTrans*. Later on, you may want to change to receiving just rotation data (Rx, Ry, Rz) by calling *SetConnexionClientMask* with a *kConnexionMaskAxisRot* mask parameter. Normally, *kConnexionMaskAll* is used to receive all types of device events.

### Declared In

*ConnexionClientAPI.h*

## 5.5 *UnregisterConnexionClient*

Unregisters your application from the driver

```
void UnregisterConnexionClient(  
    uint16_t clientID  
);
```

### Parameters

*clientID*

ID returned from a call to *RegisterConnexionClient*.

### Discussion

This function signals the driver to stop sending device messages to your application. You must call this function prior to exiting your application.

### Declared In

ConnexionClientAPI.h

## 6. Callbacks

### 6.1 *ConnexionAddedHandlerProc*

Defines the format of your device added handler callback function.

```
typedef void (*ConnexionAddedHandlerProc)(  
    unsigned int productID  
);
```

If you name your function `MyAddDeviceHandler`, you would declare it like this:

```
void MyDeviceAddedHandler(  
    unsigned int connection  
);
```

#### Parameters

*connection*

The connection object

#### Discussion

Called whenever a 3Dconnexion device is plugged into a USB port on the computer.

### 6.2 *ConnexionRemovedHandlerProc*

Defines the format of your device removed handler callback function.

```
typedef void (*ConnexionRemovedHandlerProc)(  
    unsigned int productID  
);
```

If you name your function `MyRemoveDeviceHandler`, you would declare it like this:

```
void MyDeviceRemovedHandler(  
    unsigned int connection  
);
```

#### Parameters

*connection*

The connection object

#### Discussion

Called whenever a 3Dconnexion device is unplugged from the computer.

### 6.3 ConnexionMessageHandlerProc

Defines the format of your device message handler callback function.

```
typedef void (*ConnexionMessageHandlerProc)(  
    unsigned int productID,  
    unsigned int messageType,  
    void *messageArgument  
);
```

If you name your function MyMessageHandler, you would declare it like this:

```
void TestMessageHandler(  
    unsigned int connection,  
    unsigned int messageType,  
    void *messageArgument  
);
```

#### Parameters

*connection*

The connection object

*messageType*

Identifies the message being sent by the driver

*messageArgument*

A pointer to a *ConnexionDeviceState* data structure object. See the definition of *ConnexionDeviceState* in the Data Types section

#### Discussion

Called whenever the device cap is actuated or when one of the device buttons is pressed or released.

## 7. Data Types

### 7.1 *ConnexionDeviceState*

Defines a device message structure from the driver to your application

```
typedef struct
{
    uint16_t      version;
    uint16_t      client;
    uint16_t      command;
    int16_t       param;
    int32_t       value;
    uint64_t      time;
    uint8_t       report[8];
    uint16_t      buttons8;
    int16_t       axis[6];
    uint16_t      address;
    uint32_t      buttons;
} ConnexionDeviceState, *ConnexionDeviceStatePtr;
```

#### Fields

##### *version*

Version number for this data structure format

##### *client*

The client ID of the target registered application. Device messages are broadcast to all applications that have registered with the driver. The application receiving a device message should check this field against the client ID returned during the call to *RegisterConnexionClient*. If the IDs match, then the application is the intended recipient of the message.

##### *command*

An enumerated numeric value for the command being sent by the driver. See the *Constants* section below for a list of possible values.

##### *param*

Optional parameter for the specified *command*.

##### *value*

Optional value for the specified *command*.

##### *time*

Timestamp for this message. The value is from `get_clock_uptime`.

##### *report*

Raw USB report from the device. You normally don't use this data but instead rely on the *buttons* and *axis* fields.

#### *buttons*

Reports which buttons, if any, are pressed. Buttons are assigned  $2^n$  values, i.e. 1, 2, 4, 8, 16, and so on. The *buttons* field is a sum of all the values being pressed. For example, if button 1 and button 2 are both pressed, the *buttons* field will be 3. When all buttons are released, the *buttons* field will be 0.

#### *axis*

The axis field is an array of 6 signed 16-bit integers corresponding to the 6 device axes. Data is ordered as Tx, Tz, Ty, Rx, Rz, Ry. The values reported are scaled by the driver according to the speed slider settings on the 3Dconnexion preference panel. At maximum speed, the range is -1024 to 1024. Typical range that you should optimize your application for should be -500 to 500.

#### *address*

An unsigned 16-bit value that identifies the USB port to which the 3Dconnexion device is connected. This data is useful when multiple 3dconnexion devices are hooked-up to the system. Your application will be able to differentiate which device is sending the event based on the address field.

#### *reserved2*

Not used.

### **Discussion**

You receive this structure in the *messageArgument* parameter of the *ConnexionMessageHandlerProc* callback function.

## 8. Constants

### 8.1 Device command values

These are the possible values for the command field of the ConnexionDeviceState structure.

```
enum
{
    kConnexionCmdNone           = 0,
    kConnexionCmdHandleRawData  = 1,
    kConnexionCmdHandleButtons  = 2,
    kConnexionCmdHandleAxis     = 3,
    kConnexionCmdAppSpecific    = 10
};
```

#### Constants

*kConnexionCmdNone*

No command. Currently not sent.

*kConnexionCmdHandleRawData*

Currently not sent.

*kConnexionCmdHandleButtons*

A button press or release event has occurred.

*kConnexionCmdHandleAxis*

One or more axis motions has been triggered.

*kConnexionCmdAppSpecific*

Currently not sent.

## 9. Samples

A number of example Xcode projects / source code to demonstrate the use of the framework and give a quick start to develop your own applications.

### **9.1 3DxCubeDemo**

Cocoa based OpenGL sample application that uses the model of a cube. Demonstrates a method of processing device data by polling the device status in a separate thread.

### **9.2 3DxSNAxisDemo**

Basic Cocoa application that displays the device's axis and button values.

### **9.3 3DxMultithreadedValues**

Cocoa based application that displays axis values and button state. This demo demonstrates the use of a thread to separate device threads and main/UI thread. It shows how to separate (3D Mouse) event creation and a worker thread, that for example draw a very complex geometry.

### **9.4 Connexion Client Test**

A simple Cocoa based sample, that shows the usage of the new SetConnexionHandlers() function and provides a “load” (sleep delay) to test for device events (NOT) “queueing up”.

This sample is prepared to build a sandboxed application and include the appropriate entitlement file.