

3DxInput API

Author:	Markus Bonk
Participant:	
Cc:	
Classification:	

Document history:

Version	Author	Date	Status	Comment
V0.1	Markus Bonk	13-Jan-07	Draft	In progress
V0.2	Markus Bonk	25-Jan-07	Draft	Update to match name changes
V0.3	Markus Bonk	02-Feb-07	Draft	Added TDxInfo and Period property
V0.4	Markus Bonk	02-Mar-07	Draft	Added Ziva's Getting Started and Data supplied sections.
V0.5	Markus Bonk	18-Apr-07	Draft	Removed references to singleton objects. Added LoadPreferences method to the Device object. Removed javascript sample
V0.6	Markus Bonk	01-Aug-07	Draft	Added description of standard navigation modes.
V0.7	Markus Bonk	12-Nov-07	Draft	Corrected description of Keyboard methods. Changed description of Sensor::Period Added Navigation Parameters chapter

Contents

1.	OVERVIEW	4
1.1	INTRODUCTION	4
1.2	HISTORY	4
1.3	REFERENCES.....	4
2.	GETTING STARTED	5
2.1	3D APPLICATION OPERATING MODES	5
2.2	ALTERNATE ORIENTATIONS	5
3.	NAVIGATION.....	6
3.1	AXIS ORIENTATION.....	6
3.2	3D NAVIGATION MODES.....	6
3.2.1	<i>Object Control</i>	6
3.2.2	<i>Camera Control</i>	6
3.2.3	<i>Fly – Level Horizon</i>	6
3.2.4	<i>Walk</i>	6
3.2.5	<i>Helicopter</i>	7
3.3	3D NAVIGATION CONTROL USAGE	7
3.4	2D NAVIGATION.....	7
3.4.1	<i>Paper Control</i>	8
3.4.2	<i>View Control</i>	8
4.	NAVIGATION PARAMETERS	9
4.1	LATENCY.....	9
4.1.1	<i>Start Delay</i>	9
4.1.2	<i>Propagation Delay</i>	9
4.1.3	<i>Frame Rate Lag</i>	9
4.1.4	<i>Regeneration Delay</i>	10
4.2	SPEED CONTROL.....	10
4.2.1	<i>Object Control Panning Speed Adjustment</i>	10
4.2.2	<i>2D Panning Speed Adjustment</i>	10
4.2.3	<i>Rotation Speed Adjustment</i>	10
5.	KEYS.....	11
5.1	PRE-DEFINED COMMANDS	11
5.1.1	<i>Command 31 – Fit</i>	11
5.1.2	<i>Command 30 – Toggle Configuration Panel</i>	11
6.	DATA SUPPLIED	12
6.1	HOW DATA SHOULD BE USED.....	14
6.2	DATA TUNING	17
7.	USING THE 3DXINPUT API.....	18
8.	3DXINPUT API OBJECT MODEL DIAGRAM.....	18
9.	DEVICE OBJECT.....	19
9.1	DEVICE METHODS	19
9.1.1	<i>Device::Connect</i>	19
9.1.2	<i>Device::Disconnect</i>	19
9.1.3	<i>Device::LoadPreferences</i>	20
9.2	DEVICE PROPERTIES.....	21
9.2.1	<i>Device::Keyboard</i>	21
9.2.2	<i>Device::Sensor</i>	21
9.2.3	<i>Device::Type</i>	21
9.3	DEVICE EVENTS	22
9.3.1	<i>Device::DeviceChange</i>	22
10.	KEYBOARD OBJECT	23

10.1	KEYBOARD METHODS	23
10.1.1	<i>Keyboard::IsKeyDown</i>	23
10.1.2	<i>Keyboard::IsKeyUp</i>	23
10.1.3	<i>Keyboard::GetKeyLabel</i>	24
10.1.4	<i>Keyboard::GetKeyName</i>	24
10.2	KEYBOARD PROPERTIES.....	25
10.2.1	<i>Keyboard::Keys</i>	25
10.2.2	<i>Keyboard::ProgrammableKeys</i>	25
10.2.3	<i>Keyboard::Device</i>	25
10.3	KEYBOARD EVENTS.....	26
10.3.1	<i>Keyboard::KeyDown</i>	26
10.3.2	<i>Keyboard::KeyUp</i>	26
11.	SENSOR OBJECT	28
11.1	SENSOR METHODS.....	28
11.2	SENSOR PROPERTIES	28
11.2.1	<i>Sensor::Device</i>	28
11.2.2	<i>Sensor::Rotation</i>	28
11.2.3	<i>Sensor::Translation</i>	29
11.2.4	<i>Sensor::Period</i>	29
11.3	SENSOR EVENTS.....	29
11.3.1	<i>Sensor::SensorInput</i>	29
12.	TDXINFO OBJECT	31
12.1	TDXINFO METHODS	31
12.1.1	<i>TDxInfo::RevisionNumber</i>	31
12.2	TDXINFO PROPERTIES.....	31
12.3	TDXINFO EVENTS.....	31
13.	ANGLEAXIS OBJECT.....	32
13.1	ANGLEAXIS METHODS	32
13.2	ANGLEAXIS PROPERTIES	32
13.2.1	<i>AngleAxis::X</i>	32
13.2.2	<i>AngleAxis::Y</i>	32
13.2.3	<i>AngleAxis::Z</i>	33
13.2.4	<i>AngleAxis::Angle</i>	33
14.	VECTOR3D OBJECT	34
14.1	VECTOR3D METHODS.....	34
14.2	VECTOR3D PROPERTIES	34
14.2.1	<i>Vector3D::X</i>	34
14.2.2	<i>Vector3D::Y</i>	34
14.2.3	<i>Vector3D::Z</i>	34
14.2.4	<i>Vector3D::Length</i>	35
15.	SAMPLES.....	36
15.1	CUBE3DPOLLING	36
15.2	ATLCUBE3D	36
15.3	MONITOR	36

1. Overview

1.1 Introduction

This reference guide documents the 3Dconnexion 3DxInput Application Programming Interface (API) which allows third party programmers to add support for 3D input devices and customize the data received.

1.2 History

- V0.1 Initial Draft
- V0.2 Updated to match new class names
Added description of IsKeyDown and IsKeyUp methods
Removed KeyState property
- V0.3 Added TDxInfo Object
Added description of the Sensor Period property
- V0.4 Added Getting Started and Data Supplied sections.
Added Data Tuning paragraph
- V0.5 Removed references to singleton objects.
Added LoadPreferences method to the Device object.
Removed javascript sample
- V0.6 Added description of the object and camera navigation modes and where these are useful
- V0.7 Corrected description of the Keyboard::GetKeyLabel() and Keyboard::GetKeyName() methods.
Changed the description of the Sensor::Period property
Added 2D navigation to the navigation chapter.
Added the navigations parameters chapter.
Added reference to TDx.TDxInput.dll assembly for .NET and a short note about deployment.
Added note to Device::Connect() concerning message loop requirement
Removed version in title
Added section on pre-defined key commands

1.3 References

2. 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.

2.1 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.

2.2 Alternate Orientations

Some users prefer to think of the Windows 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 Windows cursor 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.

3. Navigation

The goal of integrating a 3D mouse into an application is to provide the user with experience as if he were holding the displayed object, or the camera, in his hand. This illusion can only be upheld if the object moves simultaneously with the hand movement and in the direction the user expects.

To achieve an excellent user experience the developer needs to understand the difference between 2D and 3D mice and the parameters that affect the navigation experience.

3.1 Axis Orientation

Where discussing axes and orientations, this document will assume a right-handed coordinate system with 'Y' as the Up axis and 'Z' pointing to the observer.

3.2 3D Navigation Modes

3.2.1 Object Control

The main characteristic of object mode navigation is that the user has the impression he is holding the object in his hand. An important use for this navigation mode is in the modeling and inspection of parts and assemblies.

To create this illusion for the user, the direction that the object moves needs to be the same as the direction the user moves his hand, which is moving the devices cap. It is also important that the center of rotation is fixed relative to the object. A consequence of this mode is that the pan speed needs to be adjusted depending how far the object is from the user (see paragraph 4.2.1 below).

3.2.2 Camera Control

Camera mode navigation is characterized by the user having the impression that he is moving around in the scene he is observing. A typical use for a camera mode is exploring virtual sceneries or in first person games. This requires that the user moves and turns in the direction that the cap on the 3D mouse moves, and causes the objects displayed to move in the opposite direction to object mode described above. In camera mode the center of rotation is at the eye or view point. Because camera mode navigation reflects movement in the real world, there are a number of sub modes which have various constraints similar to those existing in the real world.

3.2.3 Fly – Level Horizon

Fly – Level Horizon mode is used to describe a camera mode navigation where the horizon is required to always remain leveled ("horizontal"). This constraint is often named 'disable rolling' but is not the same as simply disabling the roll axis on the device as the combination of the tilt and spin rotations will result in a rotation in the roll axis. Also incorrect is ignoring the roll axis inputs and applying the spin directly around the world's up axis.

Apart from the horizon constraint fly mode is the same as unconstrained camera mode, with the center of rotation located in the camera.

3.2.4 Walk

This mode is used where the height above a surface is required to remain constant whilst still allowing the user to look up and down. Walk mode has the same constraint applied to the horizon as Fly – Level Horizon mode in that the horizon must remain leveled. The translations applied to the cap of the device are considered to be in the world zx plane with the device's y-axis disabled: Pushing the cap forward moves the observer forward in the virtual plane he is standing independent of whether he happens to be looking up or down. An analogy would be that the device controls the walker's body whilst the tilt axis controls the walker's head or eyes.

3.2.5 Helicopter

As the name suggests this mode simulates a helicopter control mechanism. Similar to walk mode navigation the device's pan axes control the movement in a plane parallel to the world's zx-plane irrespective of the applied tilt.

However, unlike walk mode, the device's y-axis is used directly to control the height above the world's zx- plane. In this navigation mode pulling the devices cap up causes the height above world's zx-plane to increase, increasing the distance of the view point above the plane. Similarly, pressing the cap down causes the view point to get closer to the plane.

Not only are the device's y-translation values applied directly to the world's up-axis, the same is true for the devices spin values: These rotations act as if the device's and the world's up-axis were coincidental.

3.3 3D Navigation Control Usage

Not all navigation modes have the same importance in every application. Which navigation mode in a virtual environment feels most natural is related to how a similar task would be performed in the real world. For example, if the user has the impression that he is looking at something he would hold, then the natural navigation mode would be object mode. Similarly, if he thinks that he is in a building or scene then the most natural mode might be 'walk'.

The table below lists a number of application categories and the navigation modes we have identified as being most appropriate for the tasks that the user may perform.

	3D Navigation Modes				
Application type	Object	Camera	Fly	Walk	Helicopter
CAD / CAM	Modeling and Inspection				
Digital Context Creation	Modeling	Inspection	Animation		
Architectural	Modeling		Inspection	Inspection	
Geographic Information System					General Navigation

Application Navigation Modes

3.4 2D Navigation

2D navigation is characterized by only being able to pan and zoom and not being able to rotate at all. One might argue that a rotation about the view direction to change the layout from portrait to landscape should be allowed. However, this forgets that navigation is about continuous smooth movement; this does not apply to changing the layout format which is the 2D equivalent of changing to a different view.

The two navigation paradigms that are of interest in 2D views are what we call paper and view control.

3.4.1 Paper Control

The 'real-life' equivalent is of a user holding a piece of paper or book in his hand. In this navigation mode the viewed object or document moves, that is pans and zooms, in the same direction as the hand movement.

3.4.2 View Control

View control is the default 2d navigation paradigm for manipulating windows using the 2d mouse and the scrollbar controls. Effectively, the only difference to Paper Control is the direction that the viewed document moves and is similar to moving your view point above a document.

4. Navigation Parameters

In order to provide a good navigation experience, a number of parameters which affect the quality of the process of transforming the user's hand movement to a visual representation in the virtual 2D or 3D viewing space need to be understood and taken into consideration when adding support for a 3D input device.

4.1 Latency

There are at least four time constants which have a major affect on the quality of the navigation experience. It is neither possible to completely remove all of them nor even guarantee that they all are within certain bounds; one of the reasons for this being that the operating systems with which the majority of us work today are not real time systems. However, for the purpose of the following discussion, the effects of CPU and resource load due to other programs will be ignored.

4.1.1 Start Delay

The time between the user moving the 3D mouse from rest and when the object starts to move on the screen is what we shall call the start delay. Ideally, the application does not need any special initialization before it starts converting the 3D input data into visual feedback. This is not true for all applications: possibly a different graphical representation or viewport type needs to be created and initialized before navigation can begin. The time that this takes results in the start delay. If this delay is too long the user will let go of the device before he sees a resulting movement on the screen and assume that the input device is defective.

4.1.2 Propagation Delay

If we assume that we are looking at a very simple model which can be drawn almost instantaneously, then the time that elapses between the user changing the 3D mouse sensor displacement and the effects of the change propagating to the view is caused solely by the propagation delay. With a very short propagation delay, the user has the impression that he is directly attached to the object that he is controlling. A noticeable propagation delay is characterized by the impression that a rubber band appears somehow to be connected between the controller the object being moved.

4.1.3 Frame Rate Lag

The frame rate that an application can maintain when the user is navigating is a measure of the usability of the 3D mouse. The reason for this is that the 3D mouse delivers values that represent the translation and rotation speed of the object or view that the user is trying to control. Increasing the frame rate results in a smaller time interval and hence a smaller distance travelled since the user was last shown the result of his pushing, pulling or turning of the cap of the device. The resulting user experience due to the high frame rate is one of a smoother and more precise navigation.

Conversely, a larger time interval between consecutive frames leads to adverse effects which leave the user with a poor navigation experience: he is not sure where the object actually is on the screen as a small movement of his hand can result in a large change of the controlled objects velocity and more than likely redrawing the screen is not synchronized with single model or scene changes.

As the scene complexity increases, the frame rate that can be achieved will decrease. There would appear little alternative to reducing the amount that needs to be redrawn from one frame to the next to be able to achieve a reasonable frame rate. Depending on the application and the capabilities of for example the API, a number of strategies can be useful. **A very common strategy, often implemented by the program itself, is to degrade the visual style used to draw in the viewport dependent on the frame rate.** It can in some cases be sufficient to notify the program that degrading should begin and after the user has returned the cap to the rest position that degradation is no longer required.

4.1.4 Regeneration Delay

The regeneration delay occurs when the user puts the cap back into the zero or rest position and the application cannot be used for a period of time longer than can be accounted for by the previously mentioned latency times. One reason for this kind of program unresponsiveness is when the scene or model needs to be redrawn in a visual style more detailed than the style used during navigation.

4.2 Speed Control

4.2.1 Object Control Panning Speed Adjustment

When using the 3D mouse to hold the viewed object and move it around in 3D, the speed of panning in the screen plane needs to be adjusted depending on the distance of the view point to the object. That is, when the user moves what he is holding nearer so that he can identify more detail and to have precise 2D mouse control, the 3D panning speed needs to be reduced accordingly.

A simple speed algorithm that accounts for the viewing distance is one that adjusts the panning speed of a point in, or on the surface of the object such that independent of the size of the object, the time that the point takes to traverse the viewport window is constant for the same displacement of the 3d mouse cap. Some minimal pan speed is required as otherwise the speed will reduce to zero when the user has moved close to the referenced point.

4.2.2 2D Panning Speed Adjustment

The speed adjustment algorithm described above also applies to 2D views. Here, instead of moving the object the size, or zoom level, is changed instead of moving in the z-axis. Nonetheless the speed that a point on the object takes to traverse the viewport window needs to be adjusted according to the visible width of the 2D world.

4.2.3 Rotation Speed Adjustment

In general, the rotation speed does not need adjusting dependent on the distance of the viewed object to the user's eye point. However, experience has shown that a rotation speed that feels comfortable using object control is too fast in camera control by a factor of approximately 2.

5. Keys

3Dconnexion devices can have various numbers of keys which consist of keys with pre-assigned specific functionality and keys that can be assigned to commands by the user. Generally, the keys are numbered from 1 consecutively up to the number of keys on the device. When the user presses a key and the command associated with it cannot be executed directly by the driver, the application will be notified that it should execute the required command.

5.1 Pre-defined Commands

There are a number of commands which are predefined and should be supported by all applications.

5.1.1 Command 31 – Fit

With 'Fit' assigned to a key, the application will be notified via the `KeyDown()` keyboard event that key 31 has been pressed, i.e. that it should execute command 31 and fit the possibly no longer visible objects to the current viewport. The exact functionality associated with 'Fit' needs to be interpreted in the context of the application. Whereas in a modeling environment this may be simply centering an object on the screen, in a GIS application this might be interpreted as setting a specific zoom level and orientation.

5.1.2 Command 30 – Toggle Configuration Panel

An application is only required to support command 30 if it has a 3D mouse configuration panel. When this command is received the application should toggle the visibility state of the 3D mouse configuration or option panel.

6. Data Supplied

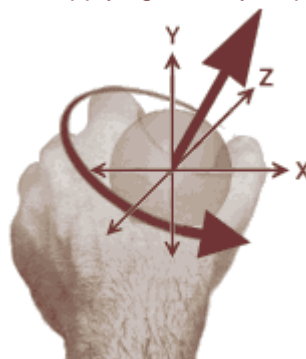
3Dconnexion devices provide full, simultaneous six-axis movement in any and all directions. The diagrams below show the orientation of Translation and Rotation axes on the sensor. Following the diagrams, there are two charts explaining the data range of each axis, and the axis 'meaning' in both 6D Navigation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

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.



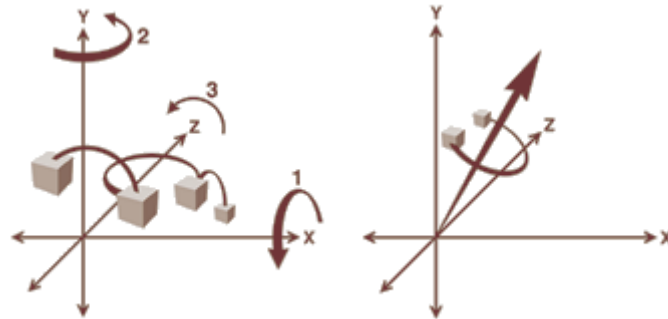
The normal range of the device axes is approximately +/- 500. The user can though, scale up or down these values in the GUI, so your application should be able to handle larger or smaller values.

The 3Dconnexion devices produce both a translation and a rotation vector simultaneously as the user pushes, pulls, or twists the device cap (as is shown in the diagram below). The translation vector is proportional to the linear displacement the user applies to the handle. The rotations returned from the device are proportional to the vector about which the user is applying a rotary displacement.



Applying a Rotation to the 3D Sensor

The translation vector is fairly easy to interpret. The three components (X, Y, and Z) of the translation vector can be applied in the same manner as similar data from the keyboard or mouse is applied to the viewing transform. The rotation vector is a different matter. Applying the rotation vector as individual parts will not give the same result as rotation about the vector (see below). If the data is applied individually the user will notice a “wobble” when performing a rotation.



Individual Rotations vs. Rotation about a Vector

6.1 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




Translation Controls

Device Value	Camera Mode	Object Mode
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 or Jumps 	Object Moves Up 

Device Value	Camera Mode	Object Mode
<p data-bbox="411 293 609 353">Translate Y-axis 0 to -MAX</p> 	<p data-bbox="724 293 1023 353">Camera Moves Down or Crouches</p> 	<p data-bbox="1106 293 1356 322">Object Moves Down</p> 
<p data-bbox="411 629 609 689">Translate X-axis 0 to MAX</p> 	<p data-bbox="740 629 995 658">Camera Moves Right</p> 	<p data-bbox="1106 629 1347 658">Object Moves Right</p> 
<p data-bbox="411 965 609 1025">Translate X-axis 0 to -MAX</p> 	<p data-bbox="746 965 989 994">Camera Moves Left</p> 	<p data-bbox="1114 965 1339 994">Object Moves Left</p> 

Rotation Control

Device Value	Camera Mode	Object Mode
<p style="text-align: center;">Rotate Z-axis 0 to MAX</p> 	<p style="text-align: center;">Left Cartwheel or Barrel Roll</p> 	<p style="text-align: center;">Spins Counterclockwise</p> 
<p style="text-align: center;">Rotate Z-axis 0 to -MAX</p> 	<p style="text-align: center;">Right Cartwheel or Barrel Roll</p> 	<p style="text-align: center;">Spins Clockwise</p> 
<p style="text-align: center;">Rotate Y-axis 0 to -MAX</p> 	<p style="text-align: center;">Spin Clockwise</p> 	<p style="text-align: center;">Spins Clockwise</p> 
<p style="text-align: center;">Rotate Y-axis 0 to MAX</p> 	<p style="text-align: center;">Spin Counterclockwise</p> 	<p style="text-align: center;">Spins Counterclockwise</p> 
<p style="text-align: center;">Rotate X-axis 0 to MAX</p> 	<p style="text-align: center;">Pitch Up or Look Up</p> 	<p style="text-align: center;">Object top spins towards you</p> 

Device Value	Camera Mode	Object Mode
<p data-bbox="427 293 588 353">Rotate X-axis 0 to -MAX</p> 	<p data-bbox="711 293 1023 322">Pitch Down or Look Down</p> 	<p data-bbox="1114 293 1345 353">Object bottom spins towards you</p> 

6.2 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 "Jet Demo" program as a reference. A proven method is having the driver (3DxWare) 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.

7. Using the 3DxInput API

The 3DxInput API is available from 3DxSoftware v3.1 and later. You can use any programming language that supports COM to use 3DxInput to add support for 3Dconnexion input devices. The COM server is implemented as the dynamic load library TDxInput.dll which is automatically registered when 3DxWare is installed.

To use 3DxInput in a C/C++ environment add the following to the stdafx.h header file `#import "progid:TDxInput.Device" embedded_idl no_namespace`. Also see the AtlCube3D sample.

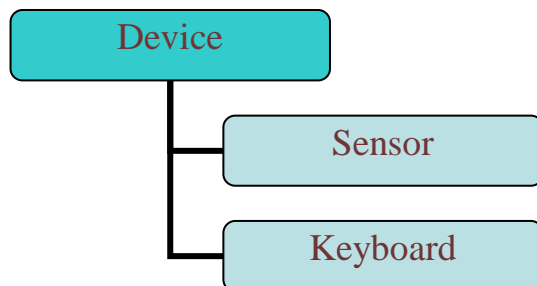
To use 3DxInput in a .NET environment a reference to the TDx.TDxInput.dll assembly needs to be set in the Visual Studio 2005 project environment. See the Microsoft Visual Studio 2005 Documentation on how to do this.

Note:

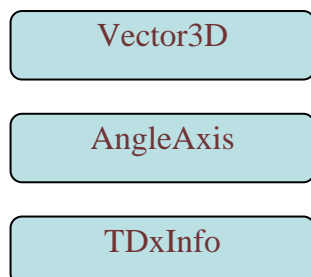
The TDx.TDxInput.dll assembly will need to be deployed with the .NET applications referencing it. The TDx.TDxInput.dll assembly may only be deployed without change to a directory local to the .NET application referencing it and must not be registered to the global assembly cache.

8. 3DxInput API Object Model Diagram

The diagram below depicts the 3Dconnexion 3DxInput API object model.



Other objects:



9. Device Object

The Device object is the base object which provides access to the 3Dconnexion device via other objects exposed in the API.

In a C++ COM application the Device object can be obtained by using CoCreateInstance. The Device methods are exposed by the ISimpleDevice interface.

9.1 Device Methods

9.1.1 Device::Connect

Description

This method connects the client to the 3dx device and enables event notification.

Syntax (COM)

```
result = ISimpleDevicePtr->Connect()
```

Return: (HRESULT) result S_OK if successful

Important:

For the device object to be able to access data from the 3D mouse the application needs to be a windows application, or more precise, is required to implement a message pump

Remarks

The Connect() method will succeed even when no device is connected to the computer or the 3DxWare driver is not running. Stopping and starting the 3DxWare driver will leave the client in the connected state, although the client's DeviceChange event handler may be executed when 3DxWare restarts.

9.1.2 Device::Disconnect

Description

This method disconnects the client from the 3dx device and disables event notification.

Syntax (COM)

```
result = ISimpleDevicePtr->Disconnect()
```

Return: (HRESULT) result S_OK if successful

Remarks

9.1.3 Device::LoadPreferences

Description

This method associates a preferences configuration with the device

Syntax (COM)

result = ISimpleDevicePtr-> LoadPreferences (preferencesName)

Input: (BSTR) preferencesName a string identifying the preferences

Return: (HRESULT) result S_OK if successful

Availability

Introduced in revision number 1.1.0

Remarks

In the current implementation of 3dxware, the preferencesName is used to identify the application receiving data from the device. A good name to use would be the name of the application or product. Configuration files are multi-device and multilingual and installed by 3dxware. To have a custom configuration file installed for your application contact 3Dconnexion.

9.2 Device Properties

9.2.1 Device::Keyboard

Description

This property returns an interface to the Keyboard object. This is a read only property.

Syntax (COM)

```
result = ISimpleDevicePtr->get_Keyboard(&IKeyboardPtr)
```

Property: IKeyboardPtr Pointer to the Keyboard interface

Return: (HRESULT) result S_OK if successful

Remarks

9.2.2 Device::Sensor

Description

This property returns an interface to the Sensor object. This is a read only property.

Syntax (COM)

```
result = ISimpleDevice->get_Sensor(&ISensorPtr)
```

Property: ISensorPtr Pointer to the Sensor interface

Return: (HRESULT) result S_OK if successful

Remarks

9.2.3 Device::Type

Description

This property returns the type of device attached to. This is a read only property.

Syntax (COM)

```
result = ISimpleDevice->get_Type(&type)
```

Property: (long) type Device attached

Return: (HRESULT) result S_OK if successful

Remarks

The type property describes the device attached: UnknownDevice = 0, SpaceNavigator = 6, SpaceExplorer = 4, SpaceTraveler = 25, SpacePilot = 29

9.3 Device Events

9.3.1 Device::DeviceChange

Description

This event is triggered when Device recognizes a new device has been attached.

Syntax (COM)

result = OnDeviceChange(long reserved)

Input: (long) reserved Reserved for future use

Return: (HRESULT) result S_OK if successful

Important:

If the event handler has been attached from managed code, it is essential to keep at least one reference to the device component to be able to receive events. When the last reference to the device runtime callable wrapper is released the garbage collection will also detach any device event handlers still attached.

Remarks

A simple method of attaching to receive event notification using COM is given by
hr = __hook(&_ISimpleDeviceEvents::DeviceChange, ISimpleDevicePtr,
&OnDeviceChange);
Here the DeviceChange event is reference via the _ISimpleDeviceEvents event interface.

10. Keyboard Object

The Keyboard object provides access to the 3Dconnexion device keyboard. The Keyboard object can be obtained by from the Device object.

10.1 Keyboard Methods

10.1.1 Keyboard::IsKeyDown

Description

This property returns the down state of a specific key on the device.

Syntax (COM)

```
result = IKeyboardPtr->IsKeyDown(key, &retval)
```

Input: (long) key	the number of key on the device
Output: (VARIANT_BOOL) retval	VARIANT_TRUE if the key is depressed
Return: (HRESULT) result	S_OK if successful

Remarks

10.1.2 Keyboard::IsKeyUp

Description

This method returns the up state of a specific key on the device.

Syntax (COM)

```
result = IKeyboardPtr->IsKeyUp(key, &retval)
```

Input: (long) key	the number of key on the device
Output: (VARIANT_BOOL) retval	VARIANT_TRUE if the key is released
Return: (HRESULT) result	S_OK if successful

Remarks

10.1.3 Keyboard::GetKeyLabel

Description

This method returns the label of a key as printed on the device.

Syntax (COM)

```
result = IKeyboardPtr-> GetKeyLabel (key, &label)
```

Input: (long) key the number of key on the device

Output: (BSTR) label label on the device

Return: (HRESULT) result S_OK if successful

Remarks

10.1.4 Keyboard::GetKeyName

Description

This method returns the name of a key on the device.

Syntax (COM)

```
result = IKeyboardPtr-> GetKeyName (key, &name)
```

Input: (long) key the number of key on the device

Output: (BSTR) name name of the key

Return: (HRESULT) result S_OK if successful

Remarks

Generally GetKeyName will return the same as GetKeyLabel. Currently the only exception is for the SpaceNavigator: The left button's name is "N1" and the right button's name is "N2". This is to be able to distinguish these keys from the "L" and "R" keys on the SpaceExplorer and SpacePilot.

10.2 Keyboard Properties

10.2.1 Keyboard::Keys

Description

This property returns the number of keys on the device. This is a read only property.

Syntax (COM)

```
result = IKeyboardPtr->get_Keys(&keys)
```

Property: (long) keys the number of keys on the device

Return: (HRESULT) result S_OK if successful

Remarks

10.2.2 Keyboard::ProgrammableKeys

Description

This property returns the number of keys on the device which may be reprogrammed. This is a read only property.

Syntax (COM)

```
result = IKeyboardPtr ->get_ProgrammableKeys(&keys)
```

Property: (long) keys The number of reprogrammable keys

Return: (HRESULT) result S_OK if successful

Remarks

10.2.3 Keyboard::Device

Description

This property returns the parent device component of the keyboard. This is a read only property.

Syntax (COM)

```
result = IKeyboardPtr ->get_Device(&ISimpleDevicePtr)
```

Property: ISimpleDevicePtr Pointer to an SimpleDevice component interface

Return: (HRESULT) result S_OK if successful

Remarks

10.3 Keyboard Events

10.3.1 Keyboard::KeyDown

Description

This event is triggered when a key on the device is pressed

Syntax (COM)

result = OnKeyDown(int keycode)

Input: (long) keycode The number of the key pressed

Return: (HRESULT) result S_OK if successful

Important:

If the event handler has been attached from managed code, it is essential to keep at least one reference to the device component to be able to receive events. When the last reference to the device runtime callable wrapper is released the garbage collection will also detach any device event handlers still attached.

Remarks

A simple method of attaching to receive event notification of the KeyDown event using COM is given by

```
hr = __hook(&_IKeyboardEvents::KeyDown, IKeyboardPtr, &OnKeyDown);
```

Here the KeyDown event is reference via the _IKeyboardEvents event interface.

10.3.2 Keyboard::KeyUp

Description

This event is triggered when a key on the device is released

Syntax (COM)

result = OnKeyUp(int keycode)

Input: (long) keycode The number of the key released

Return: (HRESULT) result S_OK if successful

Important:

If the event handler has been attached from managed code, it is essential to keep at least one reference to the device component to be able to receive events. When the last reference to the device runtime callable wrapper is released the garbage collection will also detach any device event handlers still attached.

Remarks

A simple method of attaching to receive event notification of the KeyUp event using COM is given by

```
hr = __hook(&_KeyboardEvents::KeyUp, IKeyboardPtr, &OnKeyUp);
```

Here the KeyUp event is reference via the _KeyboardEvents event interface.

11. Sensor Object

The Sensor object provides access to the 3D motion data of the 3Dconnexion device.

In a C++ COM application the Sensor object can be obtained from the Device object.

11.1 Sensor Methods

11.2 Sensor Properties

11.2.1 Sensor::Device

Description

This property returns the parent device component of the sensor. This is a read only property.

Syntax (COM)

```
result = ISensorPtr->get_Device(&ISimpleDevicePtr)
```

Property:	ISimpleDevicePtr	Pointer to an SimpleDevice component interface
-----------	------------------	--

Return:	(HRESULT) result	S_OK if successful
---------	------------------	--------------------

Remarks

11.2.2 Sensor::Rotation

Description

This property returns the rotation component of the sensor data. This is a read only property.

Syntax (COM)

```
result = ISensorPtr->get_Rotation(&IAngleAxisPtr)
```

Property:	IAngleAxisPtr	Pointer to an AngleAxis component interface
-----------	---------------	---

Return:	(HRESULT) result	S_OK if successful
---------	------------------	--------------------

Remarks

11.2.3 Sensor::Translation

Description

This property returns the translation component of the sensor data. This is a read only property.

Syntax (COM)

```
result = ISensorPtr ->get_Translation(&IVector3DPtr)
```

Property: IVector3DPtr Pointer to an Vector3D component interface.

Return: (HRESULT) result S_OK if successful

Remarks

11.2.4 Sensor::Period

Description

This property returns the ideal event frequency.

Syntax (COM)

```
result = ISensorPtr ->get_Period(&period)
```

Property: (DOUBLE) period Data time period

Return: (HRESULT) result S_OK if successful

Remarks

11.3 Sensor Events

11.3.1 Sensor::SensorInput

Description

This event is triggered when data has been received from the 3Dconnexion device.

Syntax (COM)

```
result = OnSensorInput()
```

Return: (HRESULT) result S_OK if successful

Important:

If the event handler has been attached from managed code, it is essential to keep at least one reference to the device component to be able to receive events. When the last reference to the device runtime callable wrapper is released the garbage collection will also detach any device event handlers still attached.

Remarks

A simple method of attaching to receive event notification using COM is given by
`hr = __hook(&_ISensorEvents::SensorInput, ISensorPtr, &OnSensorInput);`
Here the SensorInput event is reference via the _ISensorEvents event interface. The sensor data can be retrieved using the Translation and Rotation properties.

12. TDxInfo Object

The TDxInfo object provides general information about the component server

12.1 TDxInfo Methods

12.1.1 TDxInfo::RevisionNumber

Description

This method returns the current TDxInput revision number

Syntax (COM)

```
result = ITDxInfoPtr->RevisionNumber (&revisionNumber)
```

Input: (long) key the number of key on the device

Output: (BSTR) revisionNumber Revision as "x.y.z"

Return: (HRESULT) result S_OK if successful

Remarks

12.2 TDxInfo Properties

12.3 TDxInfo Events

13. AngleAxis Object

The AngleAxis object provides a representation for orientation in 3D space using an angle and an axis. The rotation is specified by a normalized vector and an angle around the vector. The rotation is the right-hand rule.

13.1 AngleAxis Methods

13.2 AngleAxis Properties

13.2.1 AngleAxis::X

Description

This property returns the X component of the vector.

Syntax (COM)

```
result = IAngleAxisPtr->get_X(&x)
```

Property: (double) X X component of the rotation vector

Return: (HRESULT) result S_OK if successful

Remarks

13.2.2 AngleAxis::Y

Description

This property returns the Y component of the vector.

Syntax (COM)

```
result = IAngleAxisPtr->get_Y(&y)
```

Property: (double) Y Y component of the rotation vector

Return: (HRESULT) result S_OK if successful

Remarks

13.2.3 AngleAxis::Z

Description

This property returns the Z component of the vector.

Syntax (COM)

```
result = IAngleAxisPtr->get_Z(&z)
```

Property: (double) Z Z component of the rotation vector

Return: (HRESULT) result S_OK if successful

Remarks

13.2.4 AngleAxis::Angle

Description

This property returns the Angle component of the rotation.

Syntax (COM)

```
result = IAngleAxisPtr->get_Angle(&angle)
```

Property: (double) angle Angle component of the rotation.

Return: (HRESULT) result S_OK if successful

Remarks

The angle is in arbitrary units. The angle is right handed

14. Vector3D Object

The Vector3D object provides a representation for direction in 3D space.

14.1 Vector3D Methods

14.2 Vector3D Properties

14.2.1 Vector3D::X

Description

This property returns the X component of the vector.

Syntax (COM)

```
result = IVector3DPtr->get_X(&x)
```

Property: (double) X X component of the vector

Return: (HRESULT) result S_OK if successful

Remarks

14.2.2 Vector3D::Y

Description

This property returns the Y component of the vector.

Syntax (COM)

```
result = IVector3DPtr ->get_Y(&y)
```

Property: (double) Y Y component of the vector

Return: (HRESULT) result S_OK if successful

Remarks

14.2.3 Vector3D::Z

Description

This property returns the Z component of the vector.

Syntax (COM)

```
result = IVector3DPtr ->get_Z(&z)
```

Property: (double) Z Z component of the vector

Return: (HRESULT) result S_OK if successful

Remarks

14.2.4 Vector3D::Length

Description

This property returns the length of the vector.

Syntax (COM)

```
result = IVector3DPtr ->get_Length(&length)
```

Property: (double) length The length of the vector

Return: (HRESULT) result S_OK if successful

Remarks

Setting the length to 1 normalizes the vector.

15. Samples

A number of examples are available demonstrating how to connect to the 3Dconnexion device.

15.1 Cube3DPolling

This is a simple 3D cube sample which connects to the 3Dconnexion device and uses polling to access the device data.

15.2 AtlCube3D

This is a simple 3D cube sample which connects to the 3Dconnexion device and demonstrates how to connect to the sensor and keyboard events

15.3 Monitor

The Visual Basic Monitor sample displays a simple form that outputs the motion values of the sensor.