Release 0.5, 28.04.1994 by Thomas Engel (tomi@shinto.nbg.sub.org)

2 *3D Device Tutorial*

Here I will try to give you a quick start in how to use the provided devices and how to program new drivers. It may not be perfect but I hope it will give you enough information to get the whole thing going. The servers main purpose is to allow easy access to 3D hardware and to be able to share it between many running applications. For more details about the 3DDeviceServer see the online Help system.

I had a Dream

The 3DDeviceServer is based on my private need of supporting 3D mouse hardware (and maybe datagloves in the future) in my BeakerBoy project (chemistry visualization). So many of the examples are taken from that project and my private development will focus on those devices that are important for this app.

Currently the server does only support 3D mouse devices. But the general approach could be the same for the other categories (gloves, scanners, etc.). I don't own every possible device so I would really enjoy hearing from people who own any kind of 3D equipment or know about hardware manufacturers of such devices. Maybe they have an interface spec.

264278_paste.eps ¬

Support

If you use this server or are planning to include support into your app please contact me. It won't cost you any money. I am only trying to be sure that the changes I might make in the future won't affect any available software and I will try to keep you informed when other device drivers are starting to see the light of day.
There might be some changes or additions to make those 3D classes more DriverKit fashinon to the outside.

Supporting software will be included into the 3DDeviceServer help pages.

754626_paste.eps ¬

This tutorial will focus one including current 3D drivers to existing software. The second part deals with basic concepts of writing mouse drivers buy may not be as detailed as the first one.

Using the 3D Mouse Devices

You might understand the following section better if you take a look at the example source given here. I will refer to methods that are in there. This code is the main rotation controller of my client app. It does deal with everything necessary to use 3D mouse drivers. The basic steps should be identical for every other device type.

Rotator.h ¬Rotator.m ¬

Well some known 'bugs' are still in there. I will show them later.

What do I really need

There are some things you really need when connecting to a 3D mouse.

- Your main rotation controller (or call it the mouse/device target) has to conform to the Misc3DMouseTarget protocol.
- You have to add the MiscRtMatrix object to your project. The mouse will pass its data in this object via the **transformationEvent:** message **bycopy**.
- You need to implement all the event handling methods defined in the protocol.
- You have to connect to the PDO port of the server. This port should match the "<*hostname*>/3DDeviceServer" pattern.
- You should ensure that your app will always be the target when it is the *active* app. (see later)

Well, sounds like a lot of work but it is not very hard to do.

Connecting to the Server

When working with 3D Devices you have to establish a connection to the server first.

If you take a look at the **connectTo3DMouse:** method you will find this code.

```
server3D = [NXConnection connectToName:"localhost/3DDeviceServer"];
ourServer = [server3D connectionForProxy];
[ourServer runFromAppKit];
```

This establishes a two-way PDO connection to the server. Currently the port is fixed to the localhost name. This will change in the future. My plan is to include the real host this device server is running on. Applications that are NXHost'ed to other servers should have no problems connecting to the device server that is running on the machine the user is really sitting infront of.
Once I will include this feature I will provide another example code here.

If you don't want to stop the event flow even when the user is inside a modal loop or dialog you should use **runFromAppKitWithPriority:***(10-30)* instead.

Once the connetion is set up correctly you can ask the server for the mouse object and then do the necessary inits.

```
mouse3D = [server3D mouse];
[mouse3D setProtocolForProxy:@protocol(Misc3DDeviceDriverProtocol)];
[mouse3D setTarget:self];
[mouse3D enableEvents];
[mouse3D setUseExternalSync:YES];
```

Setting the protocol enables sending the oneway messages and speeds up the delivery. The next part is the device init section. Setting the devices target to the

responsible rotation controller and starting the event delivery with extranl sync. Sync'ing events shoul be the default because the 3DKit might never keep up with the delivered events (almost impossible with 3D graphics of unlimited complexity!). So the syncing will ensure clean event passing. This may lead to the effect that the sending driver will have to swallow some events when the receiving client is not ready to accept them.

The init part should be added to the **appDidBecomeActive:** method of your apps delegate. This will ensure that when the user swapped app focus and returned to your app it willl continue to receive the events even when another program took over for the last few minutes. A **appDidHide:** might stop sending the events or what ever. No limits here.

This allows sharing a device. Future versions of the server might add the feature of locking to a certain target. But this won't affect any client code! So don't worry. There is no way for a client to prevent another app from becoming the mouse target!

Events¼or is the Answer 42

Once you have a proper connection you only need to handle the events correctly. Most apps that already deal with 3D graphics of any kind should have some methods that are similar to those of my example code. Mapping the mouse events to internal

methods should not be that hard. But there are some things one should be aware of! See later.

The first event we will receive is the (oneway void)**transformationWillStart:***sender* message. Here we have the chance to switch the drawing style of our world shape or camera model to an almost real time speed style (nice word). So the feedback will be as good as possible.

The next message is the (BOOL)**worldIsRightHanded** question. You should return the truth here. Most 3DKit views should be left-handed. So return **NO** if you are not sure.
The driver needs this information to be able to make the mouse handling intuitiv. The mouse movement should really reflect the movement of the visible object. This can only be done when the handedness of the target space is known.

Here we have reached the big problem that will show off in the next event. The transformationEvent called (oneway void)**transformationEvent:**(bycopy id)*aMatrix.* Here we get a matrix object that carries the current transformation matrix. A very simple way of how to work with it would be to simply concat it to a certain shape. This might lead to the problem that the movement won't be intuitive anymore! Sometimes it is wiser to seperate the rotation and translation transformation (like in the example code). We will leave the rotation center in the center of the object which is what you would expect normally.

Attention: As noted before there is a hidden problem (not a bug!) here. Imagine that you have a

mouse and you are looking at it. You will always look at it from the same place. You won't start rotating the whole mouse on your desk or the desk itself¼or would you ??? Anyway, how will this affect your code ?
It is simple to move the camera around the world. But transformations will *only* be correct and intuitive when the worlds camera is in the right position inside the coordinate system you will apply your transformation to!
The coordinate system has to be oriented the same way as you are to the mouse. Hmm, what does that mean? In left-handed worlds your camera has to stay on the *negative* z-axis with the world x-axis to the right and the y-axis upwards. The same is true for the right-handed world. Here the camera has to be on the *positive* z axis. This is *not* a bug of the server! Its a general problem of applying transformations to different coordinate systems.

The final step is quite simple. (oneway void)**transformationDidEnd:***sender* should return to the nice but slow display style. The mouse driver should be smart enough to send those messages with a smart timing so there won't be unnecessary redraws in the slow drawing style.

What about the Sync ?

Right. We forgot about the sync. It's quite simple. After redrawing the scene we will

send the mouse driver a **syncEvents** message. The device will send us the next event once it is available.

Without syncing we might have a typical ghost-session. Our shape would continue performing many nice but now unexpected rotations or moves depending on the event messages still in the event/method queue.
Therefore you should always use syncing when there is no need to catch every event. The drivers must ensure that syncing will produce useful events.

Advanced Features

I did not mention the **keyEvent:** method because it is not implemented inside the drivers yet. Anyway, to support them you would have to stick to a certain device because not every mouse does offer the same additional keys. SpaceMouse supports 1,2,3,4,5,6,7,8,*. Others might support none.

To solve this problem our drivers include the **realDriver** method. This will return you the driver that really does all the work. You might use this drivers advanced features and methods after checking its class. But it is not recommended to use that feature too often. It will limit your code to certain devices. For more details see the specific driver release notes.

Attention: Never try using the device you get directly from the server in another way then mention above. It is only a driver wrapper that just understands the basic device control methods.

PDO leaves many possible ways of connection control and other stuff like that. If you have the time you should include it. The basic code inside the example works fine and should cause no trouble¼even when connection problems appear. Many methods are **oneway** so they should not slow the app down even when the server got killed.

I hope this introducion is enough to get you going. If you have more questions please contact me.

Writing a new Mouse Driver

Sorry, this section is somewhat short but people who want to write a driver should be able to read and understand the two included drivers. They are well commented and should be quite simple to change. I will try to explain the basic steps here.

The single Parts

Most drivers will consist of a frontend (or inspector) and a driver. The frontend should be a subclass of the MiscSwapContentsController to be able to show its control view. The driver should be a subclass of the relating abstract driver. Currently we only have the Misc3DMouseDriver. So there is little to choose from :-)

Note: If you don't have the MiscKit object collection you should pick it form the internet archives. It includes a detailed description of the MiscSwapKit and the MiscSerialPort used in this project. It's worth the diskspace.

There is no class checking inside the app but you have to support the protocols and you need to implement all the methods need. The easiest way is to be a subclass. Right ?

Adding a new driver to the 3DDeviceServer does require adding your driver and frontend to the whole project and recompiling it. Future version will support loading bundles. (expect this for summer '94 - unless there is demand for this earlier)

The Frontend

The frontend should control all the settings a driver offers. It should not contain any real configuration code other than reading default values from the users dafaults

database and applying them through the drivers set-methods.

It should be a subclass of the MiscSwapContentsController to be swapable. I should know about the main swapView, the related button and its main contentsView. In addition it should have its own icon which represents the device it controls.
For more details on swapping see the MiscKit documentations.

The Driver...don't drink and drive !

The driver should be able to connect to the right hardware device through the right hardware port. It also should contain all the methods to control the device and its settings. This way a client might be able to directly control a special hardware (not a common case). So please don't place hardware related code into the frontend.

Take a look at the SpaceMouse driver. It does include handling a serial port, timed entrys to track activity, communication with the frontend, syncing settings made at the mouse and inside the frontend, using the target switching to update internal data and simple target events.

When you take a look at the Virtual 3D Mouse driver you find that it only has a frontend. It simply uses the plain Misc3DMouseDriver to handle the dataflow. The abstract mouse driver does most of the hardware independant mouse code. Like

setting the target, tracking syncs and event en/disabling plus the delivery of rotation & translation values.
Key-events will be included soon.

When writing a driver you have to remember that syncing should deliver useable values. For a absolut position mouse this would mean that the last sucessfully deliverd transformation will have to serve as a reference for the next event to be delivered.

It's not hard work. A mouse driver should be done within one or two days. Add another two for writing a docu and the help pages (which you should send me so I can include them). So keep the drivers coming. Presonally I have not seen a mouse comparable to the SpaceMouse as far a ease of use and professionality are concerned. SpaceBall might come close but I was not able to lay my hands on it. I would be interested in other peoples experiences.

Thank you for all the Fish...

A lot of devices are missing. Many APIs have to be hammered out yet. Software needs to support this server to make it really as useful as possible.

I have many ideas but this is not my main project so I won't spend too much time on

'useless' work. Everybody who would like to help designing device APIs is invited to join.

The server did ease my work of supporting 3D mouse devices. I hope it will be useful for you too. Although this tutorial may not be perfect it might be a place to start.