3Dconnexion Navigation Framework (Navlib)

Author:	3Dconnexion
Participant(s):	
Cc:	
Classification:	*Public documentation*

Document history summary:

Version	Author	Date	Status	Comment
0.1	3Dconnexion	2015-Feb-09	Draft	Preliminary documentation
0.2	3Dconnexion	2016-Dec-13	Draft	Updated to navlib v0.6.0
0.3	3Dconnexion	2018-Jan-30	Draft	Updated to navlib v0.7.0
0.4	3Dconnexion	2018-May-14	Draft	Added description of samples and
				Reference to Navigation3D
0.5	3Dconnexion	2018-June-15	Draft	Added Getting Started
0.6	3Dconnexion	2018-July-20	Draft	Added probe functions.
0.7	3Dconnexion	2019-Jan-08	Draft	Added missing pivot properties
0.8	3Dconnexion	2022-Aug-10	Draft	Added new model properties

Content

1. Introduction

1.1 Purpose

This document includes information on the 3Dconnexion Navigation Framework API (Navlib).

1.2 Target Audience

The target audience of this document is all software developers integrating support for 3Dconnexion products in programs with 3D visualization capabilities.

1.3 Document History

Version 0.1

- First document version. Includes preliminary information on the 3Dconnexion Navigation framework API. The status of Navlib as of document version 1.0 is “work-in-progress / experimental”.

Version 0.3

- Document changes to the matrix representation in navlib v0.7.0.
- API naming change.
- Changed SiActionNode_t to SiActionNodeEx_t.

Version 0.4

- Document changes to samples.
- Added reference to the C# Navigation3D assembly.

Version 0.5

- Document the header and library files.

Version 0.6

- Document the probe functions.

Version 0.7

- Document the pivot properties.

Version 0.8

- Document the floor plane and units-to-meters properties.
- Change document classification to “public documentation”.

1.4 References

[1] 3Dconnexion, “3DxWare SDK 3.0 for Windows”; v. 3.0.2, revision 9261; 3Dconnexion Documentation; 11-Sep-2013.

[2] UGS Corp, “input3d_action_interface.h” Documentation 2005.

2. Overview

The 3Dconnexion Navigation Framework (Navlib) is 3Dconnexion's Application Programming Interface (API) for applications currently under active development. The information included in this document is subject to change.

2.1 3Dconnexion's Input Devices

For a detailed explanation on the operation of 3Dconnexion's input device see reference [1]. Note that 3Dconnexion Navigation does not require any additional 3Dconnexion Software Development Kit (SDK) packages.

2.2 3Dconnexion Navigation Library

The 3Dconnexion Navigation Library (TDxNavlib) is a module that implements regular navigation models used with a 3D mouse (object, camera, target-camera, "helicopter") freeing the program developers from having to re-implement the control features expected by 3D mouse users.

The framework works by querying properties of the view and using these together with the 3D Mouse input data to calculate updated property values and then putting these back to the application.

Generally, the navigation library requires that the client interface implements both a getter and a setter method for each property. A property that only has a getter is, thus, read only and one with only a setter is write only. The library itself implements, with a couple of exceptions, both methods for each property.

The complete list of properties and their types that the Navigation Framework uses is defined in navlib.h in the array propertyDescription.

The 3Dconnexion Navigation Framework will be the base of 3Dconnexion's SDKs for multiple platforms.

2.3 Getting Started

The following assumes that the 3DxWare SDK is installed in <3DXWARE_SDK_DIR>.

2.3.1 C/C++ Header Files

The header files for the C/C++ navigation library are located in the <3DXWARE_SDK_DIR>\inc\navlib directory. For the compiler to find the header files add <3DXWARE_SDK_DIR>\inc to the include directory path. Generally only a single header file is required: To use the navlib add #include <navlib/navlib.h> to the source file. When compiling for C++ the navigation library uses the namespace 'navlib'.

navlib.h	the main navigation library header file. Declares the exported functions and constants.
navlib_error.h	defines the types used for error handling.
navlib_types.h	defines the types used in the navigation library.
navlib_defines.h	macro definitions.
navlib_templates.h	defines various template functions used in C++.
navlib_operators.h	defines the C++ operator overloads for the navlib types
navlib_ostream.h	defines the C++ ostream operators for the navlib types

Note: navlib_operators.h and navlib_ostream.h are not automatically included by navlib.h.

2.3.2 C/C++ Library Files

The library files for the C/C++ navigation library are located in <3DXWARE_SDK_DIR>\lib. For x64 applications link against <3DXWARE_SDK_DIR>\lib\x64\TDxNavlib.lib, for 32 bit link against <3DXWARE_SDK_DIR>\lib\x86\TDxNavlib.lib.

2.3.3 C# Assembly

For C# application development the TDx.SpaceMouse.Navigation3D.dll assembly is provided that manages the marshalling to TDxNavLib.dll and exposes a C# interface. The assembly is located in <3DXWARE_SDK_DIR>\lib\bin.

Note: The TDx.SpaceMouse.Navigation3D assembly is part of the 3Dconnexion 3DxWare driver and is installed by the driver distribution into the Global Assembly Cache.

2.3.4 Interface Probe and Logging

Copy the navlib.xml configuration file to %appdata%\3Dconnexion\3Dxware\Cfg to enable logging. The navlib will also export probe commands when the application opens a connection that can be assigned to 3DMouse buttons to test the client interface. For the commands to return meaningful results a model needs to be loaded and some part of the model needs to be selected. The results are logged in %localappdata%\3Dconnexion\3Dxware\<appname>.navlib.log. If a test fails, the command is aborted.

2.3.4.1 navlib_probeGetters command

This command will invoke all the getter functions supplied by the client. The function attempts to verify the data received such as the format and order of the matrices, left/right-handed coordinate system etc.

2.3.4.2 navlib_probeViewport command

The viewport probe will query the viewport getter functions, check for data consistency and attempt to zoom the viewport using the viewport mutator functions.

2.3.4.3 navlib_probeCameraMatrix command

The camera probe will query the camera related getter functions, attempt to navigate the camera and verify that the camera has moved to the correct position.

2.3.4.4 navlib_probeMutators command

The probe will write values to the write only properties one after the other.

2.3.4.5 navlib_probeHittest command

The probe will invoke the hit-test functions. For the probe to succeed a loaded model is required.

2.3.4.6 navlib_probeInterface command

The probe will invoke all the previous commands one after the other.

3. Usage

The 3Dconnexion Navigation Framework is implemented in file `TDxNavLib.dll` which is part of the 3Dconnexion driver installation.

3.1 3Dconnexion Navigation Framework Dependencies

The navlib depends on 3Dconnexion's driver.

3.2 Coordinate System

The framework assumes a right-handed coordinate system with the Y-axis up and the camera looking down its negative Z-Axis.

3.3 The Navlib 'C' Interface

The navlib interface consists of 4 functions.

3.3.1 NlCreate function

Creates a new navigation instance. It specifies the name of the instance and the properties that are available for querying and updating by the navigation framework.

Syntax:

long NlCreate (nlHandle_t *pnh, const char* appname, accessor_t property_accessors[], size_t accessor_count, const nlCreateOptions_t* options);

Parameters

pnh `[out] Type: nlHandle_t*`

A pointer to a handle for the new navigation instance.

appname `[in]`

The name of the application

property_accessors `[in]`

An array of accessor_t structures containing the property name, accessor and mutator functions the client exposes to the navigation instance.

accessor_count `[in]`

The number of accessor structures in the property_accessor parameter

options `[in]`

The initialization options for the connection. This may be NULL.

Return value

If the function succeeds the return value is 0 and pnh contains the handle to the navigation instance.

If the function fails the return value is not 0 and pnh contains `INVALID_NAVLIB_HANDLE`

Remarks

3.3.2 NlClose function

Closes an open navigation instance handle and destroys the navigation instance.

Syntax:

long nl_Close NlClose(nlHandle_t nh);

Parameters

nh `[in] Type: nlHandle_t`

A valid handle to an open navigation instance.

Return value

If the function succeeds the return value is zero.

If the function fails the return value is nonzero.

Remarks

3.3.3 NlReadValue function

Read the value of a property cached in the navlib
Syntax:

long NlReadValue(nlHandle_t nh, property_t name, value_t *value);

Parameters

nh `[in] Type: nlHandle_t`

A valid handle to the open navigation instance.

name `[in] Type: property_t`

The name of the property whose value is being queried.

value `[out] Type: value_t*`

A pointer to a value_t structure that contains the property value when the function returns.

Return value

If the function succeeds the return value is zero and value contains the property data.

If the function fails or the property does not exist the return value is nonzero.

Remarks

3.3.4 NlWriteValue function

Write the value for a property to the navlib instance.

Syntax:

long NlWriteValue(nlHandle_t nh, property_t name, const value_t *value);

Parameters

nh `[in] Type: nlHandle_t`

A valid handle to the open navigation instance.

name `[in] Type: property_t`

The name of the property whose value is being written in the navlib.

value `[in] Type: const value_t*`

A pointer to a value_t structure that contains the new property value.

Return value

If the function succeeds the return value is zero.

If the function fails or the property does not exist the return value is nonzero.

Remarks

3.3.5 fnGetProperty_t prototype

Prototype for the accessor function defined in the client for the navlib to get the value of the client property.

Syntax:

typedef long (cdecl *fnGetProperty_t)(const param_t param, const property_t name, value_t* value);

Parameters

param `[in] Type: param_t`

The value of the 4th member of the accessor structure passed in nl_Create.

name `[in] Type: property_t`

The name of the property whose data is being retrieved.

value `[in] Type: value_t*`

A pointer to a value_t structure that contains the property data when the function returns.

Return value

If the function succeeds the return value is zero.

If the function fails or the property does not exist the return value is nonzero.

Remarks

3.3.6 fnSetProperty_t prototype

Prototype for the mutator function defined in the client for the navlib to set the value of the client property.

Syntax:

typedef long (cdecl *fnSetProperty_t)(const param_t param, const property_t name, const value_t* value);

Parameters

param `[in] Type: param_t`

The value of the 4th member of the accessor structure passed in nl_Create.

name `[in] Type: property_t`

The name of the property whose value is to be set.

value `[in] Type: const value_t*`

A pointer to a value_t structure that contains the new property value.

Return value

If the function succeeds the return value is zero.

If the function fails or the property does not exist the return value is nonzero.

Remarks

3.4 Navlib Properties

The framework works by querying properties and using these together with the 3D Mouse input data to calculate updated property values and then putting these back to the application. Not all properties are required in every motion model that the framework implements. Dependent on the properties available, Navlib will attempt to classify which motions models are available to the user.
The Navlib queries and sets properties using the accessors and mutators supplied by the application in the accessors parameter of the NlCreate call. The application can explicitly write the value of a property using the Navlib's exported NlWriteValue function.

3.4.1 General properties

3.4.1.1 motion_k property

Specifies that a motion model is active.

Type: bool_t

The motion_k property is set to true by the navlib to notify the client that it is executing a motion model and will update the camera matrix regularly. This is useful for clients that need to run an animation loop. When the navlib has finished navigating the camera position it will set the property to false. By writing false to the motion_k property of the navlib, a client may temporarily interrupt a navigation communication and forces the Navlib to reinitialize the navigation.

3.4.1.2 active_k property

Specifies that the navigation instance is currently active.

Type: bool_t

Clients that have multiple navigation instances open need to inform the navlib which of them is the target for 3D Mouse input. They do this by writing true to the active_k property of a navigation instance.

3.4.1.3 coordinateSystem_k property

Specifies the transform from the client's coordinate system to the navlib coordinate system.

Type: matrix_t

The Navigation Library coordinate system, as previously mentioned, is Y up, X to the right and Z out of the screen. This property is queried directly after a new navigation instance is created. This allows the client to specify the other properties using the coordinate system used in the client. For the keep Y up ('Lock Horizon') algorithm to work correctly a non-identity matrix needs to be specified whenever the ground plane is not the X-Z plane.

3.4.1.4 focus_k property

Specifies that the application has keyboard focus.

Type: bool_t

Clients that run in container applications via the NLServer proxy write this property to indicate keyboard focus. This will set 3DMouse focus to the navlib connection.

3.4.1.5 transaction_k property

Specifies the navigation transaction.

Type: long

The Navigation Library can set more than one client property for a single navigation frame. For example, when navigating in an orthographic projection possibly both the view affine and extents will be modified depending on the 3DMouse input. The Navigation Library will set the transaction_k property to a value >0 at the beginning of a navigation frame and to 0 at the end. Clients that need to actively refresh the view can trigger the refresh when the value is set to 0.

3.4.1.6 frame_timing_source_k property

Specifies the source of the frame timing.

Type: long

By writing 1 to the frame.timing_source_k' property, the client application informs the Navigation Library that the client will be the source of the frame timing.

3.4.1.7 frame_time_k property

Specifies the time stamp of the animation frame in milliseconds.

Type: double

When the frame_timing_source_k property is 1, the client initiates a frame transaction by writing the frame time property.

3.4.1.8 views_front_k property

Specifies the orientation of the view designated as the front view.

Type: affine_t

The Navlib will query this value when the connection is created and use it to orientate the model to one of the ‘Front’, ‘Back’, ‘Right’, ‘Left’ etc. views in response to the respective pre-defined view commands. If the orientation is redefined by the user the client must update the value in the Navlib.

3.4.2 View properties

3.4.2.1 view_affine_k property

Specifies the matrix of the camera in the view.

Type: matrix_t

This matrix specifies the camera to world transformation of the view. That is, multiplying this matrix on the right by the position (0, 0, 0) yields the position of the camera in world coordinates. The navlib will, generally, query this matrix at the beginning of a navigation action and then set the property per frame. The frame rate that the navlib attempts to achieve is related to the ysync of the main display.

3.4.2.2 view_constructionPlane_k property

Specifies the plane equation of the construction plane as a normal and a distance (general form of the equation of a plane).

Type: plane_t

This property is used by the Navigation Library to distinguish views used for construction in an orthographic projection: typically, the top, right left etc. views. The Navigation Library assumes that when the camera’s look-at axis is parallel to the plane normal the view should not be rotated.

3.4.2.3 view_extents_k property

Specifies the orthographic extents the view in camera coordinates.

Type: box_t

The orthographic extents of the view are returned as a bounding box in camera/view coordinates. The navlib will only access this property if the view is orthographic.

3.4.2.4 view_fov_k property

Specifies the vertical field-of-view of a perspective camera/view in radians.

Type: double

3.4.2.5 view_frustum_k property

Specifies the frustum of a perspective camera/view in camera coordinates.

Type: frustum_t

The navlib uses this property to calculate the field-of-view of the perspective camera. The frustum is also used in algorithms that need to determine if the model is currently visible. The navlib will not write to this property. Instead, if necessary, the navlib will write to the view_fov_k property and leave the client to change the frustum as it wishes.

3.4.2.6 view_perspective_k property

Specifies the projection of the view/camera.

Type: bool_t

This property defaults to true. If the client does not supply a function for the navlib to query the view's projection (which it will generally do at the onset of motion), then it must write the property in the navlib if the projection is orthographic or when it changes.

3.4.2.7 view_target_k property (for internal use only).

Specifies the target constraint of the view/camera.

Type: point_t

The camera target is the point in space the camera is constrained to look at by a 'lookat' controller attached to the camera. The side effects of the controller are that panning the constrained camera will also result in a camera rotation due to the camera being constrained to keep the target position in the center of the view. Similarly panning the target will result in the camera rotating.

3.4.2.8 view_rotatable_k property

Specifies whether the view can be rotated.

Type: bool_t

This property is generally used to differentiate between orthographic 3D views and views that can only be panned and zoomed.

3.4.3 Model properties

3.4.3.1 model_extents_k property

This property defines the bounding box of the model in world coordinates.

Type: box_t.

The model extents is used by the navigation library for the zoom algorithms to fit the model to the view or canvas. It is also used in the hit-testing algorithms to determine whether the model is in front of the camera and can be hit as well as assessing the validity of a returned hit position.

3.4.3.2 model_floorPlane_k property

This property defines the plane equation of a floor, using the general form of the equation of a plane.

Type: plane_t.

The plane equation is used by the Navigation Library to determine the floor for the walk navigation mode. In the walk navigation mode, the height of the eye is fixed to 1.5m above the floor plane. The floor need not be parallel to the world ground plane. The Navigation Library assumes that the value does not change during navigation. Should it change then the application will have to write the new value to the Navigation Library.

3.4.3.3 model_unitsToMeters_k property

This property specifies the length of the model/world units in meters.

Type: double

The conversion factor is used by the Navigation Library to calculate the height above the floor in walk mode as well as the speed in the first-person motion model. The Navigation Library requires that this value does not change, and it is only queried once.

3.4.4 Selection properties

3.4.4.1 selection_affine_k property

Specifies the matrix of the selection.

Type: matrix_t

This matrix specifies the object to world transformation of the selection. That is, for row vectors multiplying this matrix on the left by the position (0, 0, 0) yields the position of the selection in world coordinates. The navlib will, generally, query this matrix at the beginning of a navigation action that involves moving the selection, and then set the property once per frame. The frame rate that the navlib attempts to achieve is related to the vsync of the main monitor, normally about 60Hz.

3.4.4.2 selection_extents_k property

This property defines the bounding box of the selection in world coordinates.

Type: box_t

The extents of the selection are returned as a bounding box in world coordinates. The navlib will only access this property if the selection_empty_k is false.

3.4.4.3 selection_empty_k property

This property defines whether the selection is empty.

Type: bool_t

When true, nothing is selected.

3.4.5 Settings properties

3.4.5.1 settings_k property

The settings_k property is indirectly used to query and write settings to the 3Dconnexion Properties UI.

Type: string_t

The property settings_k does not exist in the Navlib. To read or write a property in the application profile, the settings_k needs to be appended with "." and the name of the profile property. I.e. "settings.MoveObjects" is used to read or write the value of the "MoveObjects" property in the profile settings.

3.4.5.2 settings_changed_k property

This property defines the change revision of the profile settings.

Type: long

This property is incremented when the settings change. If the client needs to know the value of a 3Dconnexion profile setting it should re-read the corresponding value when settings_changed_k is changed.

3.4.6 Mouse pointer properties and hit testing

The navigation library includes navigation algorithms that are dependent on the size of and the distance to the model being viewed. This can be at the view center or at the position of the mouse cursor. When these algorithms are activated, the navigation library may require the client to perform hit-testing. In this case the navigation library will set up a ray and query the nearest hit.

3.4.6.1 pointer_position_k property

This property defines the position of the mouse cursor (possibly on the projection plane) in world coordinates. The property is read only.

Type: point_t.

In OpenGL the position would typically be retrieved using `gluUnProject` with winZ set to a value >0.0.

3.4.6.2 hit_lookfrom_k property

The hit_lookfrom_k property defines the origin of the ray used for hit-testing in world coordinates.

Type: point_t.

This property is set by the navlib.

3.4.6.3 hit_direction_k property

The hit_direction_k property defines the direction of the ray used for hit-testing in world coordinates.

Type: vector_t.

This property is set by the navlib.

3.4.6.4 hit_aperture_k property

The hit_aperture_k property defines the diameter of the ray used for hit-testing.

Type: float.

This property is set by the navlib. For perspective projections this is the diameter of the ray on the near clipping plane.

3.4.6.5 hit_selectionOnly_k property

The hit_selectionOnly_k property specifies whether the hit-testing is to be limited solely to the current selection set.

Type: bool_t.

This property is set by the navlib.

3.4.6.6 hit_lookAt_k property

The hit_lookAt_k property specifies the point of the model that is hit by the ray originating from the hit_lookFrom_k position.

Type: point_t

This property is queried by the navlib. The navlib will generally calculate if it is possible to hit a part of the model from the model_extents_k and selection_extents_k properties before setting up the hit-test properties and querying this property.

3.4.7 Pivot properties

In order to help the user understand the effects of rotating the 3DMouse cap a widget marking the center of rotation should be displayed.

3.4.7.1 pivot_position_k property

The pivot_position_k property specifies the center of rotation of the model in world coordinates.

Type: point_t

This property is normally set by the navlib. The application can manually override the navlib calculated pivot by writing a specific pivot position that the navlib will use until it is cleared again by the application.

3.4.7.2 pivot_visible_k property

The pivot_visible_k property specifies whether the pivot widget should be displayed.

Type: bool_t

In the default configuration this property is set by the navlib to true when the user starts to move the model and to false when the user has finished moving the model.

3.4.7.3 pivot_user_k property

The pivot_user_k property specifies whether an application specified pivot is being used.

Type: bool_t

To clear a pivot written by the application and to use the pivot algorithm in the navlib, the application writes false to this property. To override the navlib pivot algorithm the application can either write true to this property, which will cause the navlib to query the pivot position it should use, or the application can write the pivot position directly using the pivot_position_k property. The navlib's pivot algorithm continues to be overridden until the application writes false to this property.

3.4.8 Application command properties

Application commands are actions that an application exposes to the user. Normally these will be invoked from a menu or toolbar in the application. The application command extension in the navlib allows the application to expose these commands to the 3D Mouse enabling the user to assign them to 3D Mouse buttons. The commands can either be exposed as a single blob or grouped into sets, whereby only one set can be the active set – this can be useful when the application workflow limits the actions a user can perform, for example in a cad application when the user is sketching.

3.4.8.1 commands_tree_k property

This property defines a set of commands. Command sets can be considered to be button banks. The set can be either the complete list of commands that are available in the application or a single set of commands for a specific application context. The navlib will not query the application for this property. It is the responsibility of the application to update this property when commands are to be made available to the user.

Type: SiActionNode_t*

3.4.8.2 commands_activeSet_k property

In applications that have exposed multiple command sets this property needs to be written to define the command set that is active. The navlib will not query the application for this property. It is the responsibility of the application to update this property when the set of commands need to be changed. Normally this will be due to a change in application state and may correspond to a menu/toolbar change. If only a single set of commands has been defined, this property defaults to that set.

Type: string_t

3.4.8.3 commands_activeCommand_k property

When the user presses a 3D Mouse button that has been assigned an application command exposed by the commands_tree_k property, the navlib will set this property. The string will be the corresponding id passed in the commands_tree_k property. Generally, the navlib will set this property to an empty string when the corresponding button has been released.

Type: string_t

3.4.8.4 images_k property

Images can be associated with commands. These are exposed to the 3Dconnexion UI elements by updating the ‘images’ property.

Type: imagearray_t

3.5 Structures

3.5.1 box_t

Defines a box using two diagonally opposing vertices.

Syntax

```
typedef struct {
    point_t min, max;
} box_t;
```

Members

min	`Type: point_t` Position of the vertex with the lowest coordinate values.
max	`Type: point_t` Position of the vertex with the highest coordinate values.

Remarks

3.5.2 accessor_t

Specifies the client accessor and mutator functions the navlib can invoke to access the application's property values.

Syntax

```
typedef struct tagAccessor {
  property_t name;
  fnGetProperty_t fnGet;
  fnSetProperty_t fnSet;
  param_t param;
} accessor_t;
```

Members

`name`	`Type: property_t` The name of the property.
`fnGet`	`Type: fnGetProperty_t` The accessor function for the navlib to call to retrieve the value of the property.
`fnSet`	`Type: fnSetProperty_t` The mutator function for the navlib to invoke to update the value of the property.
`param`	`Type: param_t` A user defined value that the navlib will pass as the first parameter of fnGet or fnSet.

Remarks

An array of accessor_t structures is passed in the NlCreate function which creates the navigation instance. The array passed defines the interface supplied by the caller for the navlib. To limit the access to either read- or write-only, the corresponding member of the structure can be set to 0.

3.5.3 imagearray_t

Defines a list of images.

Syntax

```
typedef struct
{
  const SiImage_t *p;
  size_t count;
} imagearray_t;
```

Members

p	`Type: SiImage_t*` Array of SiImage_t instances.
count	`Type: size_t` Number of SiImage_t instances.

Remarks

3.5.4 frustum_t

Defines a frustum.

Syntax

```
typedef struct {
  double left, right, bottom, top, nearVal, farVal;
} frustum_t;
```

Members

left	`Type: double` The coordinate of the left vertical clipping plane in camera coordinate space
right	`Type: double` The coordinate of the right vertical clipping plane in camera coordinate space
bottom	`Type: double` The coordinate of the bottom horizontal clipping plane in camera coordinate space
top	`Type: double` The coordinate of the top horizontal clipping plane in camera coordinate space
nearVal	`Type: double` The distance to the near clipping plane in camera coordinate space
farVal	`Type: double` The distance to the far clipping plane in camera coordinate space

Remarks

This is the equivalent of the parameters passed in the OpenGL function glFrustum. The navlib also uses this property to calculate the field-of-views of the camera. Thus, the values passed need to be consistent with the other camera values i.e.

tan (fov_h / 2) = (right-left) / (2 * nearVal)

and

tan (fov_v / 2) = (top-bottom) / (2 *nearVal)

3.5.5 matrix_t

Specifies a 4x4 column major matrix.

Syntax

```
typedef struct {
  double m00, m01, m02, m03
        , m10, m11, m12, m13
        , m20, m21, m22, m23
        , m30, m31, m32, m33;
} matrix_t;
```

Members

`mxy`   `Type: double`

Remarks

3.5.6 nlCreateOptions_t

Specifies the navlib initialization options.

Syntax

```
typedef struct tagNlCreateOptions {
  uint32_t size;           /* sizeof (nlCreateOptions_t) */
  bool bMultiThreaded;     /* false (default) = singlethreaded */
  nlOptions_t nlOptions;   /* combination of the nlOptions */
} nlCreateOptions_t;
```

Members

`size`   `Type: uint32_t`
The size of the structure

`bMultiThreaded`   `Type: bool`
false (default) – single-threaded.

`nlOptions`   `Type: nlOptions_t`

Remarks

An nlCreateOptions_t structure is passed in the nl_Create function. Specifying bMultithreaded=false will make the navlib query and set properties in the same thread that NlCreate was invoked on.

3.5.7 nlOptions_t

Specifies navlib behavior options.

Syntax

```
typedef enum nlOptions {
  none = 0,
  nonqueued_messages = 1,
  column_vectors = 2
} nlOptions_t;
```

Remarks

3.5.8 plane_t

Defines a plane in 3D space.

Syntax

```
typedef struct {
  vector_t n;
    double d;
} plane_t;
```

Members

n	`Type: vector_t` the normal to the plane
d	`Type: double` Distance of the plane from the origin. Positive if the origin lies above the plane, negative if below.

Remarks

A point p on the plane satisfies the equation n.(p-point_t(0,0,0))+d=0;

3.5.9 point_t

Specifies a 3D position.

Syntax

```
typedef struct {
  double x, y, z;
} point_t;
```

Members

x	`Type: double` X coordinate
y	`Type: double` Y coordinate
z	`Type: double` Z coordinate

Remarks

3.5.10 SiActionNodeEx_t

The structure represents a node of the action tree for the action set.

Syntax

```
typedef struct siActionNodeEx_s
{
  uint32_t                    size;
  SiActionNodeType_t          type;
  struct siActionNode_s       *next;
  struct siActionNode_s       *children;
  const char                  *id;
  const char                  *label;
  const char                  *description;
} SiActionNodeEx_t;
```

Members

`size` `Type: uint32_t`
the size field must always be the byte size of siActionNodeEx_s

`type` `Type: siActionNodeType_t`
The type field specifies one of the following values.
SI_ACTIONSET_NODE
SI_CATEGORY_NODE
SI_ACTION_NODE

The root node (and only the root node) of the tree always has type SI_ACTIONSET_NODE. Only the leaf nodes of the tree have type SI_ACTION_NODE. All intermediate nodes have type SI_CATEGORY_NODE.

`next` `Type: siActionNode_t*`
pointer to the next node

`children` `Type: siActionNode_t*`
pointer to the child nodes

`id` `Type: const char*`
The id field specifies a string identifier for the action set, category, or action represented by the node. The field is always non-NULL and constructed from a basic set of printable ASCII characters: "A" to "Z" (27 uppercase letters), "a" to "z" (27 lowercase letters), "0" to "9" (digits), "_" (underscore character), "-" (dash character) and "." (dot or period character). This string needs to remain constant across application sessions and across application releases. The string must not change with the application language or other UI localization. The id is used by the application to identify an action.

`label` `Type: const char*`
The label field specifies a UTF8 localized/internationalized description for the action set, category, or action represented by the node. The label field can be NULL for the root and intermediate category nodes that are not explicitly presented to users. All leaf (action) and intermediate nodes containing leaf nodes have non-NULL labels. If the application only has a single action tree set, then the label of the root (context) node can also be NULL.

`description` `Type: const char*`
The description field specifies a UTF8 localized/internationalized tooltip for the action set, category, or action represented by the node. The description field can be NULL for

the root and intermediate category nodes that are not explicitly presented to users. Leaf (action) should have non-NULL description.

Remarks

A set of actions is composed of a linked list of SiActionNodeEx_t structures. Sibling nodes are linked by the next field of the structure and child nodes by the children field. The root node of the tree represents the name of the action set while the leaf nodes of the tree represent the actions that can be assigned to buttons and invoked by the user. The intermediate nodes represent categories and sub-categories for the actions. An example of this would be the menu item structure in a menu bar. The menus in the menu bar would be represented by the SiActionNodeEx_t structures with type SI_CATEGORY_NODE pointed to by each successively linked next field and the first menu item of each menu represented by the structure pointed to by their child fields (the rest of the menu items in each menu would again be linked by the next fields).

3.5.11 SiImage_t

Describes an image.

Syntax

```
typedef struct siImage_s
{
  uint32_t                size;
  SiImageType_t           type;
  const char             *id;
  union {
    struct siResource_s   resource;
    struct siImageFile_s  file;
    struct siImageData_s  image;
    struct siImageGlyph_s glyph;
  };
} SiImage_t;
```

Members

size `Type: unint32_t`
Sizeof SiImage_t.

type `Type: SiImageType_t`
Defines the type of data in SiImage_t i.e. which of the structures in the union contains the image data.

id `Type: const char*`
Application defined zero terminated string used to identify the image. To associate and image with a command, the image and command ids need to be identical.

Remarks

3.5.12 value_t

A variant structure

Syntax

```
typedef struct value {
  propertyType_t type;
```

```
    union
    {
      void *p;
      bool_t b;
      long l;
      float f;
      double d;
      point_t position;
      vector_t vector;
      plane_t plane;
      box_t box;
      frustum_t frustum;
      matrix_t matrix;
      const SiActionNode_t* pnode;
      string_t string_;
    };
  } value_t;
```

Members

type	Type: `propertyType_t` The type of data contained in the union.
b	Type: `bool_t` A 32-bit Boolean value. A value of 0 (all bits 0) indicates false.

Remarks

3.5.13 vector_t

Specifies a 3D direction.

Syntax

```
  typedef struct {
    double x, y, z;
  } vector_t;
```

Members

x	Type: `double` X value
y	Type: `double` Y value
z	Type: `double` Z value

Remarks

3.6 The Navlib C# Interface

3.6.1 Navigation3D Assembly

The Navigation3D assembly provides C# applications with a managed interface to the native navigation library. All the required marshaling between managed and unmanaged code is implemented in the Navigation3D class. To enable 3D navigation the implementer is required to instantiate a Navigation3D class with an implementation of the INavigation3D interface.

For more information on the Navigation3D assembly and implementing the INavigation3D see the 3DxTestNL sample and the documentation TDx.SpaceMouse.Navigation3D.pdf

3.7 Frame Rate and Animation

When navigating a camera with the SpaceMouse, the view needs to be continuously updated frame by frame with new scene data to give the impression of continuous movement. The Navigation Library supports two modes for controlling the frame rate of the resulting animation.

For applications that already continuously render the scene, such as games, game engines, web applications, etc., the frame_timing_source_k property should be set to 1. By setting frame_timing_source_k to 1, the client application informs the Navigation Library that it will control the animation frame rate and be the source of the timing for the frames.

The Navigation Library indicates the beginning of a SpaceMouse movement by setting the motion_k property to true. Then, before each frame is due to be drawn, the client needs to set the frame_time_k property with the new frame's time stamp. This causes the Navigation Library to calculate new frame property values from the previous values, using the frame time, 3dmouse input data and the set navigation mode.

The actual epoch used for the frame's time stamp is not of importance as all calculations are based on the time duration since the previous frame. However, the requirements for the source clock are that it must be stable, monotonic and that the frame time units are in milliseconds.

Applications that only render the scene in response to a scene, view, or model change, for example CAD applications, should leave the frame_timing_source_k property at the default value of 0. The Navigation Library will then use the monitor vsync as the timing source.

Irrespective of the value of the frame_timing_source_k property, when the Navigation Library changes properties of the camera for a frame it will first set the transaction_k property to non-zero, then the frames properties, such as the camera affine, view extents, etc. and lastly the transaction_k property back to zero to indicate that all the frame values have been set. When the client application needs to render the view it should do so when the Navigation Library sets the transaction_k property back to zero.

4. Sample Code

4.1 navlib_viewer

The MFC sample demonstrates the implementation of a viewer connecting to the navlib. The sample uses a C++ wrapper class to the navlib C interface.

Mainfrm.cpp
The CMainFrame class creates and closes the navigation connection, implements the accessors and mutators required by the navlib interface, as well as exporting the commands and images required for assigning commands to the 3DMouse buttons.

Requirements

3DxWare 10.5.6, 3DxWare SDK 4.0, Visual Studio 2015, MFC

4.2 3DxTestNL

This C# Model-View-ViewModel sample demonstrates the implementation of a 3D WPF viewer connecting to the navlib.

The INavigation3D interface is implemented by the NavigationModel partial classes in the theme based files NavigationModel.cs, ModelCallbacks.cs, Space3DCallbacks.cs, ViewCallbacks.cs, PivotCallbacks.cs and HitCallbacks.cs.

The MainExecutor class creates a NavigationModel instance and demonstrates how to export commands and images for assigning application commands to 3DMouse buttons and handle command activation events.

Requirements

3DxWare 10.5.6, 3DxWare SDK 4.0, Visual Studio 2015, C# 6.0