

3Dconnexion Navigation Library SDK

Quick guide for C++ implementation

Document history summary:

Version	Author	Date	Comment
1.0	3Dconnexion	2020-Feb-14	Initial version
1.1	3Dconnexion	2020-Mar-04	Including survey link for beta program
1.2	3Dconnexion	2020-Apr-23	Removing view_target_k from the accessors table (for internal use only)

1. In this guide, the 3DxTraceNL sample project included in the SDK is adopted as template. The sample uses C++11, however the Navigation Library can be implemented in any program that can bind to a C interface. In your project, make sure that you reference the SDK include directory, and that the linker can find the `TDxNavLib.lib` library.
2. The communication between the Navigation Library and your application is realized by means of accessors defined in the `IAccessors` interface. Create a navigation model class that implements this interface, or inherit from the `CNavigation3D` base class:

```
#include <SpaceMouse/CNavigation3D.hpp>

namespace TDx {

class CNavigationModel : public SpaceMouse::Navigation3D::CNavigation3D, private ISignals {
    typedef SpaceMouse::Navigation3D::CNavigation3D base_type;
public:
    CNavigationModel() = delete;
    ...
}
```

3. The `CNavigation3D` implementation will create a single-threaded instance, and matrices can be row- or column-major, according to the parameters passed to its constructor:

```
/// <summary>
/// Initializes a new instance of the CNavigation3D class.
/// </summary>
/// <remarks>The default is single-threaded, column major matrices</remarks>
CNavigation3D(bool multiThreaded = false, bool rowMajor = false)
    : m_enabled(false), m_pImpl(CNavlibImpl::CreateInstance(this, multiThreaded, rowMajor)) {}
```

4. The `CNavigation3D` base class also includes a specific implementation of the accessors with pre-defined values as in the table below:

Property	Description	Get accessor	Set accessor
<code>motion_k</code>	Specifies that a motion model is active	None	<code>long SetMotionFlag (bool value);</code>
<code>coordinateSystem_k</code>	Specifies the transform from the client's coordinate system to the Navlib coordinate system (<code>CNavigation3D</code> implementation: right-handed, X-right Y-up)	<code>long GetCoordinateSystem (navlib::matrix_t &affine) const;</code>	None

transaction_k	Specifies if the navigation transaction has ended	None	long SetTransaction (long value);
view_front_k	Specifies the orientation of the view designated as the front view (CNavigation3D implementation: [1 0 0, 0 1 0, 0 0 1, 0 0 0 1])	long GetFrontView (navlib::matrix_t &affine) const;	None
view_affine_k	Specifies the matrix of the camera in the view	long GetCameraMatrix (navlib::matrix_t &affine) const;	long SetCameraMatrix (const navlib::matrix_t &affine);
view_constructionPlane_k	Specifies the plane equation of the construction plane (if any) as a normal and a distance	long GetViewConstructionPlane (navlib::plane_t &plane) const;	None
view_extents_k	Specifies the orthographic extents of the view in camera coordinates	long GetViewExtents (navlib::box_t &affine) const;	long SetViewExtents (const navlib::box_t &value);
view_fov_k	Specifies the field-of-view of a perspective camera/view (in radians)	long GetViewFOV (double &fov) const;	long SetViewFOV (double fov);
view_frustum_k	Specifies the frustum of a perspective camera/view in camera coordinates	long GetViewFrustum (navlib::frustum_t &frustum) const;	long SetViewFrustum (const navlib::frustum_t &frustum);
view_perspective_k	Specifies the projection of the view/camera	long GetIsViewPerspective (navlib::bool_t &persp) const;	None
view_rotatable_k	Specifies whether the view can be rotated (CNavigation3D implementation: true)	long GetIsViewRotatable (navlib::bool_t &isRotatable) const;	None
model_extents_k	Defines the bounding box of the model in world coordinates	long GetModelExtents (navlib::box_t &extents) const;	None
selection_affine_k	Specifies the matrix of the selection	long GetSelectionTransform (navlib::matrix_t &affine) const;	long SetSelectionTransform (const navlib::matrix_t &affine);
selection_extents_k	Defines the bounding box of the selection in world coordinates	long GetSelectionExtents (navlib::box_t &extents) const;	None
selection_empty_k	Defines whether the selection is empty	long GetIsSelectionEmpty (navlib::bool_t &empty) const;	None
pointer_position_k	Defines the position of the mouse cursor on the projection plane in world coordinates	long GetPointerPosition (navlib::point_t &position) const;	None
hit_lookfrom_k	Defines the origin of the ray used for hit-testing in world coordinates	None	long SetHitLookFrom (const navlib::point_t &position);
hit_direction_k	Defines the direction of the ray used for hit-testing in world coordinates	None	long SetHitDirection (const navlib::vector_t &direction);
hit_aperture_k	Defines the diameter of the ray used for hit-testing	None	long SetHitAperture (double diameter);
hit_selectionOnly_k	Specifies whether the hit-testing is to be limited solely to the current selection set	None	long SetHitSelectionOnly (bool value);
hit_lookAt_k	Specifies the point of the model that is hit by the ray originating from the lookfrom position	long GetHitLookAt (navlib::point_t &position) const;	None
pivot_position_k	Specifies the centre of rotation of the model in world coordinates	long GetPivotPosition (navlib::point_t &position) const;	long SetPivotPosition (const navlib::point_t &position);
pivot_visible_k	Specifies whether the pivot widget should be displayed	long GetPivotVisible (navlib::bool_t &visible) const;	long SetPivotVisible (bool visible);
pivot_user_k	Specifies whether an application specified pivot is being used (CNavigation3D implementation: false)	long IsUserPivot (navlib::bool_t &userPivot) const;	None

Get accessors provide the parameters of your application 3D viewport to the Navigation Library, while *Set* accessors are used to update those parameters after the new frame computations following a 3D mouse movement. If a property is not available or not used in the application, you can notify the Navigation Library about it by putting the following error code in the corresponding *Get* accessor:

```
long GetViewConstructionPlane(navlib::plane_t &plane) const override {
    return navlib::make_result_code(navlib::navlib_errc::no_data_available);
}
```

5. Implement the Action Interface by exporting icons and commands:

```
/// <summary>
/// Exports the application commands to the 3Dconnexion Properties Configuration Utility.
/// </summary>
void CApplication3D::ExportApplicationCommands() {
    using SpaceMouse::CCategory;
    using SpaceMouse::CCommand;
    using SpaceMouse::CCommandSet;

    m_applicationCommands = {
        {ID_CLOSE, [this]() { CloseFile(); }},
        {ID_OPEN, [this]() { OpenFile(); }},
        {ID_EXIT, [this]() { Exit(); }},
        ...
    };

    // A CommandSet can also be considered to be a button bank, a menubar, or a
    // set of toolbars
    CCommandSet menuBar("Default", "Ribbon");

    // Create some categories / menus / tabs to the menu
    {
        // File menu
        CCategory menu("FileMenu", "File");
        menu.push_back(CCommand(ID_OPEN, "Open file...", "Open a 3D image file.));
        menu.push_back(CCommand(ID_CLOSE, "Close file", "Close the current 3D image file.));
        menu.push_back(CCommand(ID_EXIT, "Exit"));
        menuBar.push_back(std::move(menu));
    }

    ...

    // Add the command set to the commands available for assigning to 3DMouse buttons
    m_navigationModel.AddCommandSet(menuBar);

    // Activate the command set
    m_navigationModel.ActiveCommands = menuBar.Id;
}

/// <summary>
/// Exports the images for the commands to the 3Dconnexion Properties Configuration Utility.
/// </summary>
void CApplication3D::ExportCommandImages() {

    // Images can be exported from three different sources:
    // - an image file from the hard-disk, by specifying the index (in case of multi-image file)
    // - a resource file from the hard-disk, by specifying resource type and index
    // - an image buffer
    // All the formats that can be loaded by Gdiplus::Bitmap::FromStream() (including
    // recognizable SVG formats) are allowed

    namespace fs = std::experimental::filesystem;

    std::vector<CImage> images = {
        CImage::FromFile(fs::canonical("images/about.png").generic_u8string(), 0, ID_ABOUT),

```

```

        CImage::FromResource("c:/windows/system32/ieframe.dll", "#216", "#2", 12, ID_OPEN),
        CImage::FromResource("c:/windows/system32/ieframe.dll", "#216", "#2", 10, ID_EXIT),
        CImage::FromFile(fs::canonical("images/close.png").generic_u8string(), 0, ID_CLOSE),
        ...
    };

    m_navigationModel.AddImages(images);
}

```

6. In your main application, enable the navigation as soon as your 3D viewport is instantiated by calling the corresponding function of the navigation model class:

```

/// <summary>
/// Initializes the navigation model instance.
/// </summary>
void CApplication3D::Enable3DNavigation() {
    // Set the hint/title for the '3Dconnexion Properties' Utility.
    m_navigationModel.Profile = YOUR_PROGRAM_NAME_GOES_HERE;

    std::error_code ec;

    // Enable input from / output to the Navigation3D controller.
    m_navigationModel.EnableNavigation(true, ec);
    if (ec) {
        // something failed
        return;
    }

    try {
        ExportCommandImages();

        ExportApplicationCommands();
    } catch (const std::exception &) {
        // something unexpected happened
    }
}

```

7. If your application implements an animation loop (which is not the case for the 3DxTraceNL sample), you can synchronize the Navigation Library output with it by setting the property `frame_timing_source_k` to `TimingSource::Application` just after enabling the navigation, and the property `frame_time_k` to `std::chrono::high_resolution_clock::now()` at the beginning of your application animation loop.
8. You're done!

For any additional information, or for enter in contact with us, please visit the dedicated Software Developer section on our website: <http://www.3Dconnexion.com/>.

Please help us in improving the quality and the functionalities of this SDK by participating to our short online survey:

https://forms.office.com/Pages/ResponsePage.aspx?id=6D6W52Acf0uhoFh_dK3cFIYF4WLi9B5OuUqnL_UKq6hUMUhBSFo4RjBXQ1paVlpUOVowNjFLQk41TS4u