

OPENNI MIGRATION GUIDE

OpenNI/NiTE 2 Migration Guide - Transitioning from OpenNI/NiTE 1.5 to OpenNI/NiTE 2

Document Version 1.1

April 2013

Disclaimer and Proprietary Information Notice

The information contained in this document is subject to change without notice and does not represent

a commitment by PrimeSense Ltd. PrimeSense Ltd.

and its subsidiaries make no warranty of any kind with regard to this material,

including, but not limited to implied warranties of merchantability

and fitness for a particular purpose whether arising out of law, custom, conduct or otherwise.

While the information contained herein is assumed to be accurate, PrimeSense Ltd. assumes no responsibility

for any errors or omissions

contained herein, and assumes no liability for special, direct, indirect

or consequential damage, losses, costs, charges, claims, demands, fees or

expenses, of any nature or kind, which are incurred in connection with the

furnishing, performance or use of this material.

This document contains proprietary information, which is protected by U.S. and international copyright laws.

All rights reserved. No part of this document may be reproduced, photocopied or translated into another language

without the prior written consent of PrimeSense Ltd.

License Notice

OpenNI is written and distributed under the Apache License, which means that its source code is freely-distributed and available to the general public.

You may obtain a copy of the License at:<http://www.apache.org/licenses/LICENSE-2.0>[1]

NiTE is a proprietary software item of PrimeSense and is written and licensed under the NiTE License terms, which might be amended from time to time. You should have received a copy of the NiTE License along with NiTE.

The latest version of the NiTE License can be accessed

at:<http://www.primesense.com/solutions/nite-middleware/nite-licensing-terms/>[2]

Trademarks

PrimeSense, the PrimeSense logo, Natural Interaction, OpenNI, and NiTE are trademarks and registered trademarks of PrimeSense Ltd. Other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such.

1 Introduction

1.1 Scope

This migration guide is targeted at developers using OpenNI versions that came before OpenNI 2, with particular emphasis on those developers using OpenNI1.5.2 who are interested in transitioning to OpenNI 2. This guide is designed as both an aid to porting existing applications from the old API to the new API, as well as to enable experienced OpenNI developers to learn and understand the concepts that have changed in this new API.

This guide provides a general overview to all programming features that have changed. Included also is a lookup table of all classes and structures in OpenNI 1.5, showing the equivalent functionality in OpenNI 2.

1.2 Related Documentation

[1] [OpenNI 2 Programmer Guide, PrimeSense.](#)

[2] [NiTE 2 Programmer Guide, PrimeSense.](#)

1.3 Support

The first line of support for OpenNI/NiTE developers is the OpenNI.org web site. There you will find a wealth of development resources, including ready-made sample solutions and a large and lively community of OpenNI/NiTE developers. See:

<http://www.OpenNI.org>[3]

If you have any questions or require assistance that you have not been able to obtain at OpenNI.org, please contact PrimeSense customer support at:

Support@primesense.com[4]

2 Overview

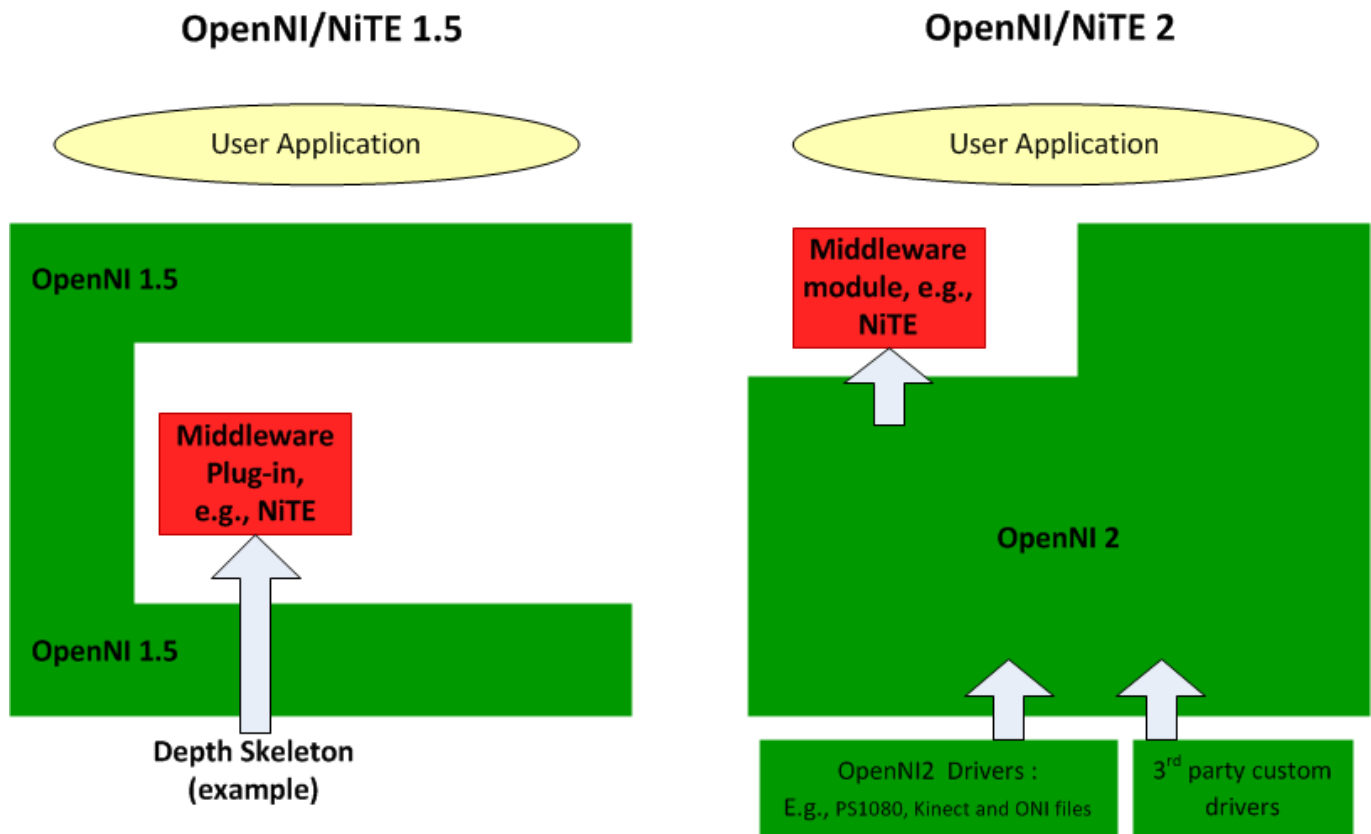
2.1 New OpenNI Design Philosophy

OpenNI 2 represents a major change in the underlying design philosophy of OpenNI. A careful analysis of the previous OpenNI API, version 1.5, revealed that it has many features that are rarely or never used by the developer community. When designing OpenNI 2, PrimeSense has endeavored to greatly reduce the complexity of the API. Features have been redesigned around basic functionality that was of most interest to the developer community. Of especial importance is that OpenNI 2 vastly simplifies the interface for communicating with depth sensors.

2.2 Simplified Middleware Interface

The plug-in architecture of the original OpenNI has been removed completely. OpenNI 2 is now strictly an API for communicating with sources of depth and image information via underlying drivers. PrimeSense NiTE middleware algorithms for interpreting depth information are now available as a standalone middleware package on top of OpenNI, complete with their own API. Previously, the API was through OpenNI only and you included NiTE functionality via plug-ins.

The following high level diagram compares the OpenNI/NiTE 2 architecture with the OpenNI/NiTE1.5 architecture.



[5]

Figure 2-1: Comparison of OpenNI/NiTE 2 architecture with OpenNI/NiTE1.5

NiTE 2 API is summarized in *Chapter ?10*, and is described in detail in the *NiTE 2 Programmer's Guide*.

PrimeSense is not aware of any third party middleware that actually made use of the provided OpenNI plug-In interfaces. All known third party middleware simply had been built to run on top of OpenNI. By simplifying the underlying interfaces that middleware developers are actually using, it is hoped that third party middleware providers will find it easier to implement their products on OpenNI.

2.3 Simplified Data Types

OpenNI 1.5 had a wide variety of complex data types. For example, depth maps were wrapped in metadata. This made it more complicated to work with many types of data that, in their raw form, are simply arrays. OpenNI 2 achieves the following:

- Unifies the data representations for IR, RGB, and depth data
- Provides access to the underlying array
- Eliminates unused or irrelevant metadata
- OpenNI 2 allows for the possibility of not needing to double buffer incoming data

OpenNI 2 solves the double buffering problem by performing implicit double buffering. Due to the way that frames were handled in OpenNI 1.5, the underlying SDK was always forced to double buffer all data coming in from the sensor. This had implementation issues, complexity problems and performance problems. The data type simplification in OpenNI 2 allows for the possibility of not needing to double buffer incoming data. However, if desired it is still possible to double-buffer. OpenNI 2 allocates one or more *user buffers* and a *work buffer*.

2.4 Transition from Data Centric to Device Centric

The overall design of the API can now be described as device centric, rather than data centric. The concepts of Production Nodes, Production Graphs, Generators, Capabilities, and other such data centric concepts, have been eliminated in favor of a much simpler model that provides simple and direct access to the underlying devices and the data they produce. The functionality provided by OpenNI 2 is generally the same as that provided by OpenNI 1.5.2, but the complicated metaphors for accessing that functionality are gone.

2.5 Easier to Learn and Understand

It is expected that the new API will be much easier for programmers to learn and begin using. OpenNI 1.5.2 had nearly one hundred classes, as well as over one hundred supporting data structures and enumerations. In OpenNI 2, this number has been reduced to roughly a dozen, with another dozen or so supporting data types. The core functionality of the API can be understood by learning about just four central classes:

```
openni::OpenNI
openni::Device
openni::VideoStream
openni::VideoFrameRef
```

Unfortunately, this redesign has required us to break backward compatibility with OpenNI 1.5.2. The decision to do this was not made lightly; however, it was deemed necessary in order to achieve the design goals of the new API.

2.6 Event Driven Depth Access

In OpenNI 1.5 and before, depth was accessed by placing a loop around a blocking function that provided new frames as they arrive. Until a new frame arrived the thread would be blocked. OpenNI 2 still has this functionality, but it also provides callback functions for event driven depth reading. These callback functions are invoked on sensors becoming available (further sensors can be connected to the host system while OpenNI is running), and on depth becoming available.

3 Summary of New Classes

| | |
|----------------|--|
| openni::OpenNI | Provides the entry point and overall context for the library, and provides access to devices, and to events related to devices entering and leaving the system. Replaces: OpenNI 1.5 Context class |
| openni::Device | Provides the connection to a device, for configuring it and obtaining video streams (implemented as VideoStream objects).A Device represents either: <i>physical hardware device</i> that is producing actual streams of data- or <i>file device</i> that contains a recording taken from a physical device.For each device there is one stream each for color, IR, and depth. Replaces: OpenNI 1.5 Device class, the Depth, IR, and Image Generator classes, an all production nodes |

| | |
|-------------------------|---|
| openni::VideoStream | Provides the means to start data generation, read individual frames, obtain information about the streams, and configure the streams. Replaces: OpenNI 1.5 DepthGenerator, IRGenerator, and ImageGenerator as the direct source of data from the sensor. |
| openni::VideoFrameRef | Encapsulates all data relevant to a single frame of data, regardless of its type, IR, Image, and Depth data Replaces: data structures for IR data, image data, and depth data in OpenNI 1.5 |
| openni::Recorder | Captures the output of one or more streams in an ONI file. |
| openni::PlaybackControl | Provides access to specific recording functionality, e.g., speed change and looping. |
| nite::UserTracker | Provides all functionality related to scene segmentation, skeleton tracking, pose detection, and user tracking. |
| nite::HandTracker | Provides all functionality related to hand tracking, including gesture detection |

4 Overall Context - The OpenNI Class

4.1 Introduction

The OpenNI class provides the entry point and overall context for the library. The OpenNI class also provides access to devices, and to events related to devices entering and leaving the system. See the *OpenNI 2 Programmer's Guide* for a complete introduction to using the OpenNI Class.

4.2 Replaces the Context class

The openni::OpenNI class provides the same overall functionality as the Context class did in OpenNI1.5. The openni::OpenNI class provides a top level area where the overall state of the API is stored. Any OpenNI 2 based application will start by running the initialization function in this class.

In OpenNI 1.5, a complex system of Production Nodes, Generators, and Production Trees/Chains was used to access depth data. In OpenNI 2 this has been simplified to a model where various hardware sensors are represented by objects of Device class. Devices provide streams of data, which in turn are composed of sequential FrameRefs (single video frames). The OpenNI class is responsible for initializing the actual hardware drivers and making physical devices accessible by Device objects.

Note that openni::OpenNI is implemented as a collection of static functions. Unlike the Context class in OpenNI 1.5, it does not need to be instantiated. Simply call the initialize() function to make the API ready to use.

4.3 Device Events (New Functionality)

OpenNI 2 now provides the following device events:

- Device added to the system
- Device deleted from the system
- Device reconfigured

These events are implemented using an OpenNI::Listener class and various callback registration functions. See the *OpenNI 2 Programmer's Guide* for details. The EventBasedRead sample

provided with OpenNI 2 provides example code showing how to use this functionality.

4.4 Miscellaneous Functionality

The various error types implemented in OpenNI 1.5 via the EnumerationErrors class have been replaced with simple return codes. Translation of these codes into human readable strings is handled by `openni::OpenNI`. This replaces the `Xn::EnumerationErrors` class from OpenNI 1.5.

OpenNI 1.5 provided a specific `xn::Version` class and `XnVersion` structure to deal with API version information. In OpenNI 2, the `OniVersion` structure is provided to store the API version. This information is accessed via a simple call to the `openni::OpenNI::Version()` function. This replaces the `xn::Version` class and the `XnVersion` structure from OpenNI 1.5.

5 The OpenNI Device Class

5.1 Introduction

In OpenNI 1.5, data was produced by `Production Nodes`. There was a mechanism whereby individual `Production Nodes` could specify dependency on each other, but in principal there was (for example) no difference between a `Depth Generator` node that was providing data from a camera, and a `Hands Generator` node creating hand points from that same data. The end application was not supposed to care whether the hand points it was using were generated from a depth map, or some other data source.

The old approach had a certain symmetry in how data was accessed, but it ignored the underlying reality that in all real cases the raw data produced by a hardware device is either a RGB video stream, an IR stream, or a Depth stream. All other data is produced directly from these streams by middleware.

OpenNI 2 has switched to a metaphor that much more closely mimics the real life situation. In doing so, we were able to reduce the complexity of interacting with hardware, the data it produces, and the middleware that acts on that hardware.

The base of the new hierarchy is the `Device` class. A `Device` represents either:

- *physical hardware device* that is producing actual streams of data

- or -

- *file device* that contains a recording taken from a physical device.

The `Device` class is used to connect to a device, configure it, and to obtain video streams (implemented as `VideoStream` objects). For each device there is one stream each for Color, IR, and Depth.

The `Device` class in OpenNI 2 replaces the `Device` class in OpenNI 1.5, as well as replacing the `Depth`, `IR`, and `Image Generator` classes.

Please see the *OpenNI 2 Programmer's Guide* for a complete introduction to using the `Device` class.

5.2 Device Specific Capabilities

In OpenNI 1.5, there was a general `Capability` class that was intended to represent the

capability of a generator in an abstract sense. Any feature or data source, whether a device or algorithm or some other data source was supposed to be represented by some sort of capability. A long list of possible capabilities was defined.

The FrameSync and AlternativeViewPoint mechanisms in OpenNI 1.5 have been replaced by a simpler system in OpenNI 2. OpenNI 2 uses 'DepthColorSync' and 'ImageRegistration' to more closely follow how real devices are actually configured. These represent underlying flags that are set in firmware or hardware on a device wide basis

The Alternative Viewpoint Capability is now referred to as Image Registration

5.3 File Devices

In OpenNI 1.5, the playback of recorded data was handled by an explicit player object and the use of virtual mock Nodes to produce the data. OpenNI 2 changes this to a system where a recording is specified in place of a physical device when the device is actually opened. The caller to the Device::open() function provides a parameter specifying whether it is a sensor device or a file device. If a recording is specified, after the initial call to the Device::open() function, a recording is in every way identical to the output from a physical device.

An openni::PlaybackControl class is also defined to control additional capabilities specific to file devices. The abilities to seek within a recording, loop the recording, and control the playback speed are all provided.

Recordings are made with the openni::Recorder class. See *chapter 8* for more information on the Recorder and PlaybackControl classes.

6 The OpenNI VideoStream Class

6.1 Introduction

The VideoStream Class replaces the DepthGenerator, IRGenerator, and ImageGenerator as the direct source of data from the sensor. It provides the means to start data generation, read individual frames, obtain information about the streams, and configure the streams. It also encapsulates the functionality formerly contained in the CroppingCapability, MirrorCapability classes, and some of the functionality of the GeneralIntCapability class.

6.2 Changes to Access Approach

Since the Generator classes have been eliminated, data is obtained directly from devices. To create a video stream, simply create a VideoStream object, and call the openni::VideoStream::create() function, passing in a valid Device object that will supply the data.

Once initialized, there are two options for obtaining data: a polling loop, and event driven access.

Polling is most similar to the functionality of the WaitXUpdateX() family in OpenNI 1.5. To perform polling, call the openni::VideoStream::start() function on a properly initialized VideoStream object. (A properly initialized VideoStream object is where a device has been opened, a stream has been associated with the device, and the start command has been given. Then, call the openni::VideoStream::readframe() function. This function will block until a new frame of data is read.

To access data in an event driven manner, you need to implement a class that extends the

`openni::VideoStream::Listener` class, and write a callback function to handle each new data frame. See the *OpenNI 2 Programmer's Guide* for more information.

6.3 Getting Stream Information

The `openni::VideoStream::getSensorInfo()` function is provided to obtain a `SensorInfo` object for an initialized video stream. This object allows enumeration of available video modes supported by the sensor.

The `openni::VideoMode` object encapsulates the resolution, frame rate, and pixel format of a given `VideoMode`. `Openni::VideoStream::getVideoMode()` will provide the current `VideoMode` setting for a given stream.

Functions are also provided by the `VideoStream` class that enable you to check resolution, and obtain minimum and maximum values for a depth stream.

6.4 Configuring Streams

To change the configuration of a stream, use the `OpenNI::VideoStream::setVideoMode()` function. A valid `VideoMode` should first be obtained from the list contained in a `SensorInfo` object associated with a `VideoStream`, and then passed in using `setVideoMode()`.

6.5 Mirroring Data

Instead of a `MirroringCapability` mirroring is now simply a flag set at the `VideoStream` level. Use the `VideoStream::setMirroringEnabled()` function to turn it on and off.

6.6 Cropping

Instead of a `CroppingCapability` cropping is now set at the `VideoStream` level. This is done directly with the `VideoStream::setCropping()` function. The parameters to be passed are no longer encapsulated in a `CroppingObj` object as they were in OpenNI 1.5 — they are now simply a collection of 4 integers.

6.7 Other Properties

The general `IntObj` capabilities have been replaced with simple get and set functions for integer based sensor settings. Use of these should be rare, however, since all of the commonly used properties (i.e., mirroring, cropping, resolution, and others) are encapsulated by other functions rather than manipulated directly.

7 The OpenNI VideoFrameRef Class

7.1 Introduction

In OpenNI 1.5.x, data was stored in a complex hierarchy of classes. There were separate classes for IR, Image, and Depth data, and a class hierarchy that was several layers deep. All of this overhead greatly obscured access to data that, at its heart, is simply a two dimensional array of pixel values.

In OpenNI 2 this complexity has been reduced by using a single `VideoFrameRef` class to encapsulate all data relevant to a single frame of data, regardless of its type. Metadata has been reduced to the minimum required to work with the frames.

7.2 Accessing frame data

The `VideoFrameRef` class includes the `VideoFrameRef::getData()` function that returns a pointer

directly to the underlying frame data. Use the `VideoFrameRef::getSensorType()` function to determine the type of data contained [Depth, IR or Color] if required.

7.3 Metadata

Functions are provided to access the following metadata properties of a frame of data:

- Data size
- Type of Sensor used to create data
- Timestamp of the frame
- Video Mode of the data (resolution and frame rate)
- Frame index (a number assigned to each frame sequentially)
- Width of frame in pixels
- Height of frame in pixels
- Cropping settings
- Stride of the array containing the data

8 OpenNI Recordings

8.1 Recorder Class

All recordings are now taken by a Recorder class. The basic purpose of the recorder class is to capture the output of one or more streams in an ONI file. To make a recording, simply give the `openni::Recorder` class a list of valid streams that you would like to record from, the name of a file to record to, and then call the `start()` function. Once you are finished recording, call the `stop()` function.

The resulting ONI file can later be used to create a device that will behave almost exactly as a physical device would.

8.2 PlaybackControl Class

It is possible to perform certain operations with a recording that cannot be done with a regular physical device. OpenNI 2 provides functions encapsulated in the `PlaybackControl` class to perform these operations. These functions simplify the use of the `Device` class for physical devices. A `PlaybackControl` object is instantiated and attached to a device file, and can then be used to access specific recording functionality, e.g., speed change and looping.

The functionality provided includes seeking within a recording, determining how many frames a recording contains, changing playback speed, and looping playback.

9 OpenNI Classes no longer Exposed

9.1 Introduction

Several classes and data structures were provided in OpenNI 1.5 that were not directly related to the central purpose of the API. These included data structures for handling points, planes, bounding boxes, etc. They also included miscellaneous structures for abstracting operating system functionality, as well as various other [helper classes] not directly related to reading depth information. These items have been removed from the API in this revision. In some cases, they are retained behind the scenes for implementation of the API, in other cases they have been removed entirely.

OpenNI developers who need any of these OpenNI classes that are no longer exposed, i.e., those who wish to extend or change the code of OpenNI itself, can find them in the PSCommon folder in the OpenNI source code.

10 Functionality Moved to NiTE 2

10.1 Overview

In OpenNI 1.5, a specific set of interfaces was provided for *gesture detection*, *skeleton tracking*, *hand point detection*, and *user detection*. Binary middleware plugins could then provide algorithms to implement these specific interfaces. PrimeSense NiTE middleware was one such implementation.

It was, however, found that few developers actually used this framework, so in OpenNI 2 the approach has been completely changed. Any attempt at standardizing the interface between middleware and applications has been abandoned. As a result, OpenNI 2 includes no interfaces for higher level middleware.

NiTE is still provided, with all the same functionality. It is now a standalone package, with its own API. Thus there are now two separate installers, one for OpenNI and one for NiTE. See the *NiTE 2 Programmer's Guide* for more information.

10.2 Full Body Tracking

10.2.1 Introduction

All functionality related to Scene Segmentation, Skeleton Tracking, Pose Detection, and User Tracking is now handled by a single **User Tracker API** `ni::UserTrackerFrameRef`. Each frame of User Tracker output is stored in an object of type `UserTrackerFrameRef`, which contains all scene segmentation, skeleton, and pose data. Data specific to a given user is stored in an object of type `UserData`. `UserData` is obtainable via accessor functions `ni::get()` in `UserTrackerFrameRef`.

10.2.2 Scene Segmentation

The UserTracker now provides an opened and working device that provides depth data via a call to `ni::UserTracker::create()`, and scene segmentation begins immediately. The `UserMap` class is used to store user segmentation data.

Step by step instructions on how to do this in code are given in the *NiTE 2 Programmer's Guide*. Alternatively, see the sample entitled `ni::impleUserTracker` for an easy to follow example of how to use all of the major User Tracker API functions. Scene segmentation now also includes a floor plane calculation.

10.2.3 Skeleton Tracking

Skeleton tracking has been incorporated into the User Tracker.

Starting the Skeleton for a given user now requires a single function call:
`ni::UserTracker::startSkeletonTracking()`

Once tracking has started, skeleton data is available as a `ni::Skeleton` object, obtained from `UserData` by calling `ni::UserData::getSkeleton`.

The Skeleton class itself simply provides access to a collection of joints. Position and orientation data for each joint is stored as a `nite::SkeletonJoint` object, which can be obtained by using `nite::Skeleton::getJoint()`. Possible types for `SkeletonJoint` are enumerated in `nite::JointType`. Joint orientation is now stored using the more standard Quaternions, instead of transformation matrices.

The Skeleton class also provides access to calibration status data via `nite::Skeleton::getState()`. Possible calibration states, including a few informative error states, are enumerated in `nite::SkeletonState`.

10.2.4 Pose Detection

Explicit Pose detection is now available by calling `nite::UserTracker::startPoseDetection()`. The list of available poses is enumerated in `nite::PoseTypes`.

10.3 Hand Tracking

Hand tracking is now performed via the `HandTracker` API. This API also includes gesture detection calls.

The `HandTracker` needs to be given a working device with the `nite::HandTracker::create()` call, and then hand tracking can be started by calling `nite::HandTracker::startHandTracking()`. All hand points and gestures detected in a given frame are encapsulated in `HandTrackerFrameRef` objects. The actual `HandPoint` data is stored as objects of type `HandData`, and gestures recognized are stored as type `GestureData`.

An enumeration of available gesture types is available in `nite::GestureTypes`.

Step by step instructions on how to do this in code are given in the *NiTE 2 Programmer's Guide*. Alternatively, see the sample entitled **`nimpleHandTrackerL/strong> for an easy to follow example of how to use all of the major Hand Tracker API functions.`**

10.4 Event Driven Programming

Both the `User Tracker` and `Hand Tracker` have associated listener class types that can be used to assign callback functions. Every new frame of data generates an event that calls these listeners. The `UserTracker` and `HandTracker` now completely encapsulate the link between `NiTE` and `OpenNI` (other than the initial function call to `create()`). It is therefore now possible to write a completely event driven application with `NiTE`. There is no need to set up a polling loop to read sensor data — that is all handled behind the scenes. Each frame of output from either the `UserTracker` or `HandTracker` provides a direct link to the depth frame that it was generated from, easing the process of combining `NiTE` output with raw depth data.

10.5 Misc

`NiTE 2` includes classes for representing points, planes, bounding boxes, quaternions, and arrays.

In `NiTE 2`, quaternions replace transformation matrices when manipulating skeleton joints.

11 Obsolete Concepts

11.1 Introduction

The architecture changes in `OpenNI 2` have caused a number of specific concepts to be no longer necessary. As a result, there are a number of classes and structures in `OpenNI 1.5` that have no equivalent in `OpenNI 2`. This chapter discusses each group of items that have been

removed, and the reasons behind these decisions.

11.2 Mock Generators

Mock Generators were used to implement the recording system in OpenNI 1.5 and to implement third party depth drivers. In OpenNI 2, recordings are created and simulated at the Device level. There is no longer any need for a special class of items to play back the data.

11.3 Queries

The elimination of Production Nodes and Production Chains has rendered the `xn::Query` object obsolete.

11.4 Modules

OpenNI 1.5 provided a large number of classes devoted to implementing its plug-in architecture. OpenNI plug-ins were referred to as `Modules`. Since this architecture no longer exists, none of these items are required.

11.5 Licensing Infrastructure

OpenNI 1.5 provided a key-value based licensing infrastructure for middleware developers to use with their code. Since OpenNI no longer directly supports middleware plug-ins, there is no reason to leave in the licensing functionality. Middleware developers are of course free to develop their own licensing mechanisms for use with their standalone middleware packages. The elimination of a licensing mechanism means that there is no longer any need for the `xn::License` class.

11.6 Script Nodes

Script nodes were a little used Production Node type. No equivalent for them has been provided in OpenNI 2.

11.7 Log Objects

OpenNI 1.5 provided a complex logging mechanism for errors. Since OpenNI 2 simplifies error reporting the log objects have been rendered unnecessary. Logging status codes in whatever form the application developer finds convenient should now be very easy to implement in the application itself.

11.8 Audio

OpenNI 1.5 provided support for a proprietary audio format via OpenNI. The only functionality provided by this audio system was a simple recording of PCM formatted sounds, with the expectation that audio would be recorded or analyzed using third party middleware. It was decided that the much more common USB-UAC (Universal Audio Class) would provide an easier to use interface for a wider number of tools. For this reason, the OpenNI 1.5 audio streams have been eliminated in favor of embracing the UAC standard. This prompted the removal of the following items from the API:

`AudioGenerator`

`AudioMetaData`

`WaveOutputMode`

11.9 Abstract Data Types

11.9.1 Introduction

OpenNI 1.5 had an extensive hierarchy of data types. For example, there were three different types of video frames (Depth, IR, RGB). Each of these types of video had its own Generator class to create the data, and its own metadata class to hold the data. The various generators had a `MapGenerator` ancestor class to abstract functionality common to each, which in turn had a

Generator ancestor, itself derived from a ProductionNode.

The simplification in OpenNI 2 has eliminated most of this hierarchy. OpenNI 2 factors out the NiTE middleware and now creates separate objects for devices and video streams. This has eliminated much of the common functionality. Merging the various video types into a single VideoFrameRef type provided further simplification. The end result is a completely flat class hierarchy. This has caused the complete elimination of many classes that served only to provide abstraction.

11.9.2 Capabilities

The concept of a generic Capability has been eliminated. In its place, various objects have simple get/set functions to activate appropriate functionality. Mirror and Cropping capabilities are now found in the VideoStream class. ImageRegistration and FrameSync capabilities are now managed from the Device class. The general Int capabilities have been replaced with get property and set property functions in either VideoStream or Device, depending on whether the setting in question affects a single stream or the entire device.

11.9.3 Production Nodes

The attempt at unifying middleware and hardware data sources has been abandoned. This has made the concept of a Production Node no longer necessary. Equivalent functionality to specific Product Node types has been retained in the VideoStream and Device classes, but the abstract hierarchy above the specific types is now gone.

11.9.4 Generators

Generators, including the specific types such as MapGenerators, have been eliminated. The Map types have all been retained as VideoStreams. Audio is now handled via UAC. The abstract hierarchy that attempted to unify these various types has been eliminated in favor of simply using VideoStreams for everything with a type code to indicate the data type.

11.9.5 MetaData

The various specific MetaData types (IRMetaData, ImageMetaData, DepthMetaData) are all handled by the openni::VideoFrameRef class. The hierarchy of abstract types (i.e., MapMetaData) has been eliminated.

Appendix A Index of OpenNI 1.5 – 2.0 Equivalents

This appendix provides a list of all classes and structures in OpenNI 1.5, along with a brief description of where to find the equivalent functionality in OpenNI 2. See the relevant chapters of this guide, the **OpenNI 2 Programmer's Guide**, or the HTML based SDK reference for additional details.

The overall structure of the two versions is different enough that a true one-to-one mapping of functions is not practical. This table is primarily intended to help you find the right area of the documentation to get the functionality you are looking for. In particular, please do not simply replace the code in the left column with the code in the right column – that will not yield a functioning program.

The table is listed alphabetically by OpenNI 1.5.2 item name. See the object in the second column for the equivalent OpenNI 2 functionality. The third column gives the section or chapter of this document where the relevant concept is discussed.

Table A-1: Index of OpenNI 1.5 – 2 Equivalents

| OpenNI 1.5.2 Class/Struct | OpenNI 2 Equivalent | See |
|------------------------------------|--|--------|
| xn::AlternativeViewPointCapability | openni::Device:isImageRegistrationSupported(), openni::Device:getImageRegistrationMode(), openni::Device:setImageRegistrationMode() | 5.2 |
| xn::AntiFlickerCapability | openni::Device:getProperty(), openni::Device:setProperty() | 5.2 |
| xn::AudioGenerator | Obsolete Concept □Audio | 11.8 |
| xn::AudioMetaData | Obsolete Concept □Audio | 11.8 |
| xn::Capability | Obsolete Concept □Abstract Data Types, See specific classes derived from Capability for their location in OpenNI 2 | 11.9.2 |
| xn::Codec | | |
| xn::Context | openni::OpenNI, openni::OpenNI:initialize() | 4.2 |
| xn::CroppingCapability | openni::Stream:getCropping(), openni::Stream:setCropping(), openni::Stream:resetCropping(), openni::Stream:isCroppingSupported(), openni::FrameRef:isCroppingEnabled() | 6.6 |
| xn::DepthGenerator | openni::VideoStream | 6 |
| xn::Device | openni::Device | 5 |
| xn::DeviceIdentificationCapability | openni::Device:getDeviceInfo() | 5 |
| xn::EnumerationErrors | OniStatus, openni::OpenNI:getExtendedError() | 4.4 |
| xn::EnumerationErrors::iterator | OniStatus, openni::OpenNI:getExtendedError() | 4.4 |
| xn::ErrorStateCapability | OniStatus, openni::OpenNI:getExtendedError() | 4.4 |
| xn::ExtensionModule | Obsolete Concept □Modules | 11.4 |
| xn::FrameSyncCapability | openni::Device:enableDepthColorSync(), openni::Device:disableDepthColorSync() | 5.2 |
| xn::GeneralIntCapability | openni::Device:getProperty(), openni::Device:setProperty(), openni::VideoStream:getProperty(), openni::VideoStream:setProperty() | 5.2 |
| xn::Generator | Obsolete Concept □Abstract Data Types, See specific classes derived from Generator for their location in OpenNI 2 | 11.9.4 |
| xn::GestureGenerator | nite::HandTracker, nite::GestureData | 10.3 |
| xn::HandsGenerator | nite::HandTracker, nite::HandData | 10.3 |
| xn::HandTouchingFOVEdgeCapability | nite::HandData::isTouchingFov() | 10.3 |
| xn::ImageGenerator | openni::VideoStream | 6 |
| xn::ImageMetaData | openni::VideoFrameRef | 7 |
| xn::IRGenerator | openni::VideoStream | 6 |
| xn::IRMetaData | openni::VideoFrameRef | 7 |
| xn::MapGenerator | Obsolete Concept □Abstract Data Types, All specific map generator functionality is available in openni::VideoStream | 11.9.4 |
| xn::MapMetaData | Obsolete Concept-Abstract Data Types, All specific MapMetaData functionality is available in openni::VideoFrameRef | 11.9.5 |
| xn::MirrorCapability | openni::VideoStream:getMirroringEnabled(), openni::VideoStream:setMirroringEnabled() | 6.5 |
| xn::MockAudioGenerator | Obsolete Concept - Mock Nodes | 11.2 |
| xn::MockDepthGenerator | Obsolete Concept - Mock Nodes | 11.2 |
| xn::MockImageGenerator | Obsolete Concept - Mock Nodes | 11.2 |
| xn::MockIRGenerator | Obsolete Concept - Mock Nodes | ?11.2 |
| xn::MockRawGenerator | Obsolete Concept - Mock Nodes | 11.2 |
| xn::Module | Obsolete Concept - Modules | 11.4 |
| xn::ModuleAlternativeViewPoint | Obsolete Concept □Modules | 11.4 |
| xn::ModuleAntiFlickerInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleAudioGenerator | Obsolete Concept - Modules | 11.4 |

| OpenNI 1.5.2 Class/Struct | OpenNI 2 Equivalent | See |
|--|--|--------|
| xn::ModuleCodec | Obsolete Concept - Modules | 11.4 |
| xn::ModuleCroppingInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleDepthGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleDevice | Obsolete Concept - Modules | 11.4 |
| xn::ModuleDeviceIdentification | Obsolete Concept - Modules | 11.4 |
| xn::ModuleErrorStateInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleExportedProductionNode | Obsolete Concept - Modules | 11.4 |
| xn::ModuleExtendedSerializationInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleFrameSyncInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleGeneralIntInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleGestureGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleHandsGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleHandTouchingFOVEdgeInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleImageGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleIRGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleLockAwareInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleMapGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleMirrorInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleNodeNotifications | Obsolete Concept - Modules | 11.4 |
| xn::ModulePlayer | Obsolete Concept - Modules | 11.4 |
| xn::ModulePoseDetectionInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleProductionNode | Obsolete Concept - Modules | 11.4 |
| xn::ModuleRecorder | Obsolete Concept - Modules | 11.4 |
| xn::ModuleSceneAnalyzer | Obsolete Concept - Modules | 11.4 |
| xn::ModuleScriptNode | Obsolete Concept - Modules | 11.4 |
| xn::ModuleSkeletonInterface | Obsolete Concept - Modules | 11.4 |
| xn::ModuleUserGenerator | Obsolete Concept - Modules | 11.4 |
| xn::ModuleUserPositionInterface | Obsolete Concept - Modules | 11.4 |
| xn::NodeInfo | Obsolete Concept - Abstract Data Types | 11.9 |
| xn::NodeInfoList | Obsolete Concept - Abstract Data Types | 11.9 |
| xn::NodeInfoList::iterator | Obsolete Concept - Abstract Data Types | 11.9 |
| xn::NodeWrapper | Obsolete Concept - Abstract Data Types | 11.9 |
| xn::OutputMetaData | Obsolete Concept - Abstract Data Types | 11.9.5 |
| xn::Player | openni::Device::open("<FILENAME>"), openni::PlaybackControl | 5.3 |
| xn::PoseDetectionCapability | nite::UserTracker, nite::PoseData | 10.3 |
| xn::ProductionNode | Obsolete Concept - Abstract Data Types | 11.9.3 |
| xn::Query | Obsolete Concept - Query | 11.3 |
| xn::Recorder | openni::Recorder | 8 |
| xn::Resolution | openni::VideoMode | 6.4 |
| xn::SceneAnalyzer | nite::UserTracker | 10.2 |
| xn::SceneMetaData | nite::UserMap | 10.2.2 |
| xn::ScriptNode | Obsolete Concept - ScriptNode | 11.6 |
| xn::SkeletonCapability | nite::UserTracker, nite::Skeleton | 10.3 |
| xn::StateChangedCallbackTranslator | openni::OpenNI::Listener::onDeviceStateChanged() | 4.3 |
| xn::UserGenerator | nite::UserTracker | 10.2 |
| xn::UserPositionCapability | nite::UserData::getCenterOfMass() | 10.2 |
| xn::Version | openni::OpenNI::getVersion() | 4.4 |
| xn::DepthMetaData | openni::VideoFrameRef | 7 |
| XnArray< T > | Essentially unchanged from OpenNI 1.5 | N/A |
| XnAudioMetaData | Obsolete Concept - Audio | 11.8 |
| XnAutoCSLocker | No longer exposed | 9.1 |

| OpenNI 1.5.2 Class/Struct | OpenNI 2 Equivalent | See |
|---|--|-------------------------|
| XnAutoMutexLocker | No longer exposed | 9.1 |
| XnBaseNode | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnBitSet | No longer exposed | 9.1 |
| XnBoundingBox3D | No longer exposed in OpenNI, nite::BoundingBox | 9.1 |
| XnCallback | openni::OpenNI::Listener, openni::Stream::Listener | 4.3 |
| XnCropping | Data now stored in members of the OpenNI::VideoStream class | 6.6 |
| XnDepthMetaData | openni::VideoFrameRef | 7 |
| XnDumpWriter | No longer exposed | 9.1 |
| XnDumpWriterFileHandle | No longer exposed | 9.1 |
| XnErrorCodeData | OniStatus, openni::OpenNI::getExtendedError() | 4.4 |
| XnEvent | Events are now simply callback function calls from openni::OpenNI and openni::VideoStream | 4.3 |
| XnEventInterface | openni::OpenNI::Listener, openni::Stream::Listener | 4.3 |
| XnFieldOfView | Data now stored as members of openni::VideoStream | 6.3 |
| XnGeneralBuffer | No longer exposed | 9.1 |
| XnHash | No longer exposed | 9.1 |
| XnHash::ConstIterator | No longer exposed | 9.1 |
| XnHash::ConstIterator | No longer exposed | 9.1 |
| XnHash::Iterator | No longer exposed | 9.1 |
| XnImageMetaData | openni::VideoFrameRef | 7 |
| XnIRMetaData | openni::VideoFrameRef | 7 |
| XnLicense | Obsolete Concept - Licensing | 11.6 |
| XnList | No longer exposed | 9.1 |
| XnList::ConstIterator | No longer exposed | 9.1 |
| XnList::ConstIterator | No longer exposed | 9.1 |
| XnList::Iterator | No longer exposed | 9.1 |
| XnLogEntry | Obsolete Concept - Logging | 11.7 |
| XnLogger | Obsolete Concept - Logging | 11.7 |
| XnLogWriter | Obsolete Concept - Logging | 11.7 |
| XnLogWriterBase | Obsolete Concept - Logging | 11.7 |
| XnMapMetaData | Obsolete Concept - Abstract Data Types | 11.9.5 |
| XnMapOutputMode | Obsolete Concept - Abstract Data Types | 11.9.5 |
| XnMatrix3X3 | No longer exposed in OpenNI, nite::Quaternion | 9.1, 10.2.3, 10.5 |
| XnModuleAlternativeViewPointInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleAntiFlickerInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleAudioGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleCodecInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleCroppingInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleDepthGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleDeviceIdentificationInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleDeviceInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleErrorStateInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleExportedProductionNodeInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleExtendedSerializationInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleFrameSyncInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleGeneralIntInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleGestureGeneratorInterface | Obsolete Concept - Modules | 11.4 |

| OpenNI 1.5.2 Class/Struct | OpenNI 2 Equivalent | See |
|--|---|--------|
| XnModuleHandsGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleHandTouchingFOVEdgeCapabilityInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleImageGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleIRGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleLockAwareInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleMapGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleMirrorInterface | Obsolete Concept - Modules | 11.4 |
| XnModulePlayerInterface | Obsolete Concept - Modules | 11.4 |
| XnModulePoseDetectionCapabilityInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleProductionNodeInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleRecorderInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleSceneAnalyzerInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleScriptNodeInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleSkeletonCapabilityInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleUserGeneratorInterface | Obsolete Concept - Modules | 11.4 |
| XnModuleUserPositionCapabilityInterface | Obsolete Concept - Modules | 11.4 |
| XnNode | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnNodeAllocator | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnNodeInfoListIterator | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnNodeNotifications | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnOpenNIModuleInterface | Obsolete Concept - Modules | 11.4 |
| XnOSEvent | No longer exposed | 9.1 |
| xnOSInfo | No longer exposed | 9.1 |
| XnOutputMetaData | Obsolete Concept - Abstract Data Types | 11.9.5 |
| XnPlane3D | No longer exposed in OpenNI, nite::Plane | 9.1 |
| XnPlayerInputStreamInterface | Obsolete Concept - Abstract Data Types | 11.9 |
| XnProductionNodeDescription | Obsolete Concept - Abstract Data Types | 11.9.3 |
| XnQueue | No longer exposed | 9.1 |
| XnRecorderOutputStreamInterface | Obsolete Concept - Abstract Data Types | 11.9 |
| XnRGB24Pixel | OniRGB888Pixel | N/A |
| XnSceneMetaData | nite::UserMap | 10.2.2 |
| XnSkeletonJointOrientation | nite::SkeletonJoint::getOrientation() | 10.2.3 |
| XnSkeletonJointPosition | nite::SkeletonJoint::getPosition() | 10.2.3 |
| XnSkeletonJointTransformation | Orientations are now stored in NiTE as quaternions | 10.2.3 |
| XnStack | No longer exposed | 9.1 |
| XnStringsKeyManager | No longer exposed | 9.1 |
| XnStringsKeyTranslator | No longer exposed | 9.1 |
| XnSupportedPixelFormat | openni::SensorInfo:getSupportedVideoModes() | 6.3 |
| XnThreadSafeQueue | No longer exposed | 9.1 |
| XnUInt32XYPair | No longer exposed | 9.1 |
| XnUSBConfigDescriptorHolder | This data type is now simply stored as a string | N/A |
| XnUSBDeviceDescriptorHolder | This data type is now simply stored as a string | N/A |
| XnUSBInterfaceDescriptorHolder | This data type is now simply stored as a string | N/A |
| XnUSBStringDescriptor | This data type is now simply stored as a string | N/A |
| XnVector3D | No longer exposed | 9.1 |
| XnVersion | OniVersion, openni::OpenNI:getVersion() | 4.4 |
| XnWaveOutputMode | Obsolete Concept - Audio | 11.8 |
| XnYUV422DoublePixel | OniYUV422DoublePixel, other than name change, this structure is basically unchanged | N/A |

Endnotes:

<http://www.apache.org/licenses/LICENSE-2.0>: <http://www.apache.org/licenses/LICENSE-2.0>

<http://www.primesense.com/solutions/nite-middleware/nite-licensing-terms/>:

<http://www.primesense.com/solutions/nite-middleware/nite-licensing-terms/>

<http://www.OpenNI.org>: <http://www.openni.org/>

Support@primesense.com: <mailto:support@primesense.com>

[Image]:

<http://www.openni.org/openni-migration-guide/comparison-betw-openni-1-5-and-2-0-docver1-1-3/>