



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[OpenNI Overview](#)

Getting Started

Installing OpenNI SDK

Installing OpenNI SDK on Windows

Double-click the provided msi file to install the SDK on your Windows machine.

The installation performs the following:

- Copies the SDK to the target directory (default is C:\Program Files\OpenNI2 or C:\Program Files (x86)\OpenNI2)
- Installs a USB driver to be used with OpenNI-compliant devices
- Defines environment variables to be used when developing OpenNI applications (see [Visual Studio](#))

Installing OpenNI SDK on Linux

1. Extract the tarball to a directory of your choice
2. Go into this directory and run the install script:

```
./install.sh
```

The installation creates udev rules which will allow usage of OpenNI-compliant USB devices without root privileges.

Samples

OpenNI SDK arrives with pre-compiled samples that can be run immediately after installation.

Under the installation directory, go to the Samples/Bin directory and run any of the samples there. Note that some samples have a graphical interface and may require a more powerfull graphic accelerator.

Creating new project that uses OpenNI

Visual Studio

1. Open a new project or an existing one with which you want to use OpenNI
2. In the Visual Studio menu, open the Project menu and choose Project Properties.
3. In the C/C++ section, under the "General" node, select "Additional Include Directories" and add "\$(OPENNI2_INCLUDE)" (if you use the 32-bit version) or "\$(OPENNI2_INCLUDE64)" (if you use the 64-bit version). These are environment variables that point to the location of the OpenNI Include directory. (The defaults are C:\Program Files\OpenNI2\Include or C:\Program Files (x86)\OpenNI2\Include)
4. In the Linker section, under the "General" node, select "Additional Library Directories" and add "\$(OPENNI2_LIB)" (if you use the 32-bit version) or "\$(OPENNI2_LIB64)" (if you use the 64-bit version). These are environment variables that point to the location of the OpenNI Lib directory. (The defaults are C:\Program Files\OpenNI2\Lib or C:\Program Files (x86)\OpenNI2\Lib)
5. In the Linker section, under the input node, select "Additional Dependencies" and add OpenNI2.lib or OpenNI2.lib
6. Ensure that you add the Additional Include and Library directories to both your Release and Debug configurations.
7. Copy all the files from OpenNI's redistributable directory (see environment variable "\$(OPENNI2_REDIST)" or "\$(OPENNI2_REDIST64)") to your working directory. (The defaults are C:\Program Files\OpenNI2\Redist or C:\Program Files (x86)\OpenNI2\Redist). Be aware that when you run from command line, the working directory is the directory where the executable can be found, and when you run from Visual Studio the default directory is where the project file (.vcproj, .vcxproj) can be found.

Note:

You may ask Visual Studio to change working directory when debugging to the directory where the executable is by

chaning "Project Properties" -> "Debugging" -> "Working Directory" to "\$(TargetDir)". Note that this setting is not kept as part of the project settings, but on a per-user, per-configuration basis.

GCC / GNU Make

In the following section, refers to the directory to where OpenNI SDK was extracted. Note that the installation does not define such an environment variable. Either define it yourself or use the full path.

1. Add the SDK Include directory, \$OPENNI_DIR/Include, to your include path (-I)
2. Copy the files from the Redist directory, \$OPENNI_DIR/Redist, to your execution directory
3. Add the execution directory to your lib path (-L)
4. Add libOpenNI2 to your library list (-l)
5. It is highly suggested to also add the "-Wl,-rpath ./" to your linkage command. Otherwise, the runtime linker will not find the libOpenNI.so file when you run your application. (default Linux behavior is to look for shared objects only in /lib and /usr/lib).

Writing an Application

- Your code should include **OpenNI.h** header file.
- The entire C++ API is available under the `openni` namespace.
- Be sure to call **openni::OpenNI::initialize()**, to make sure all drivers are loaded If no drivers are found, this function will fail. If it does, you can get some basic log by calling **openni::OpenNI::getExtendedError()** (which returns a string) Note that usually this method fails because OpenNI redist files weren't copied to the working directory.
- When closing your application, call **openni::OpenNI::shutdown()**, to allow OpenNI to close properly (unload drivers and such).
- Open a device using its URI. You can get a list of available devices using **openni::OpenNI::enumerateDevices()**. Enumeration returns an array of **openni::DeviceInfo** objects, which include (among other things) the device URI. If you don't care which device

to use, you can specify `openni::ANY_DEVICE` as the URI. (to work with .oni files, use the path to the file as its URI)

- Create a video stream by specifying the device and the sensor.
 - Be sure to `destroy` the stream and `close` the device when you're done.
-



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[OpenNI Overview](#)

C++ API Conventions

Classes vs. Structs

In general, classes are preferred, along with `getX()` / `setX()` methods. This allows extending the API in the future without breaking compatibility. In cases of "super-primitives" (like [openni::RGB888Pixel](#)), structs are used.

Return Values from Methods

- All getters return the value as their return value, and cannot fail, for example: [openni::VideoFrameRef::getWidth\(\)](#). If the value is a complex type, it will be returned as const reference, for example: [openni::Device::getDeviceInfo\(\)](#). If a specific property doesn't exist for the specific object, a meaningful value will be returned.
- stop / destroy / close methods does not have return values, for example: [openni::Device::close\(\)](#).
- All other methods return [openni::Status](#), for example: [openni::VideoStream::create\(\)](#).

Arguments to Methods

- Output arguments, as well as in/out arguments are always passed as a pointer, for example: `openni::VideoStream::readFrame()`.
- Input arguments:
 - Primitives are passed by value, for example: `openni::VideoMode::setPixelFormat()`.
 - Complex types are passed as const reference, for example: `openni::VideoStream::create()`.
 - In the rare cases where the object identity matters, and not it's content, it will be passed as a pointer, for example: `openni::OpenNI::addListener()`.

Object Lifetime

Some objects have their lifetime governed by the user. For example, a **openni::VideoStream** is created using the **create()** method and destroyed by calling **destroy()**. In those cases, it is important that they get destroyed in the right order. For example, if you create a **openni::VideoStream** on a specific **openni::Device**, don't close the device before destroying that video stream. In the same manner, all **openni::VideoFrameRef** objects should be released before destroying the stream, and **all** OpenNI objects should be destroyed before calling **openni::OpenNI::shutdown()**.

Other objects are not created by the user, but are actually members of other objects. For example, calling **openni::Device::getDeviceInfo()** returns a reference to a **openni::DeviceInfo** object. Note that once the **openni::Device** object that was used gets closed, the **openni::DeviceInfo** reference is no longer valid. Do not try to use it to access methods of the **openni::DeviceInfo** object.

Arrays

Some methods return an array of items. We created a very simple template class, [openni::Array](#), which only holds the pointer to the underlying C array and the number of elements in that array. Once the array has been released, the elements are no longer accessible. Do not keep a pointer to a specific item in the array and use it after releasing it.

Properties and Commands

OpenNI has a mechanism for generic properties and commands, both for devices and for streams. Commands are used by calling `openni::Device::invoke()` or `openni::VideoStream::invoke()` and Properties can be read via `openni::Device::getProperty()` and `openni::VideoStream::getProperty()` and written via `openni::Device::setProperty()` and `openni::VideoStream::setProperty()`.

Though both might use both primitives and complex types, generally, properties are used to encapsulate a logical data unit (even though complex) while commands are used in the following cases:

- The action requires some arguments (that are not part of the data unit).
- Obtaining a data unit via `getProperty()` have side effects.
- The operation is expensive enough that you want to communicate it to the user.
- The action has side effects.
- The data unit might be changed over time without an outside request.

Legal Stuff & Acknowledgments

OpenNI 2.X

Copyright (c) 2012 PrimeSense Ltd.

This product is licensed under the Apache License, Version 2.0 (the "License");

You should have received a copy of the Apache License along with OpenNI 2.X. If not, see: <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The following open source components are included in OpenNI 2.X:

1. LibJPEG, 6b of 27-Mar-1998

Source code can be downloaded from: <http://libjpeg.sourceforge.net/>

This software is based in part on the work of the Independent JPEG Group. OpenNI 2.X source code also contains the source code of libjpeg, as well as its license terms. An online copy can be found at:

<https://github.com/OpenNI/OpenNI2/blob/master/ThirdParty/LibJPEG/RE>

This license can be found at:

<https://github.com/aseprite/aseprite/blob/master/docs/licenses/libjpeg->

[LICENSE.txt](#)

We've made some minor modifications to the source code.

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy. This software is copyright (C) 1991-1998, Thomas G. Lane. All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions: (1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation. (2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group". (3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind. These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us. Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software". We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.
ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA. ansi2knr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2knr.c for full details.) However, since ansi2knr.c is not needed as part of any program generated from

the IJG code, this does not limit you more than the foregoing paragraphs do. The Unix configuration script "configure" was produced with GNU Autoconf. It is copyright by the Free Software Foundation but is freely distributable. The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable. It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has been removed from the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.) So far as we are aware, there are no patent restrictions on the remaining code. The IJG distribution formerly included code to read and write GIF files. To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders. We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

2. LibUSB, 1.0.9, under GNU Lesser General Public License, v 2.1 (or any later version)

OpenNI 2.X uses a modified version of libusb, which was modified by PrimeSense on 2012.

The modified source code is distributed along with OpenNI 2.X source code and can be found in the following path:

ThirdParty/PSCommon/XnLib/ThirdParty/libusb-1.0.9-Android

Copyright (C) 2007-2008 Daniel Drake <dsd@gentoo.org> Copyright (c) 2001 Johannes Erdösfeld <johannes@erdfelt.com>

This library is free software; you can redistribute it and/or modify it

under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Full text of the license is available below.

3. GLUT 3.7.x

Some of the samples arrive with GLUT binaries and headers for Windows. Those are under the following copyright:

This license can be found at:

<http://user.xmission.com/~nate/glut/README-win32.txt>

The OpenGL Utility Toolkit (GLUT) distribution contains source code published in a book titled "Programming OpenGL for the X Window System" (ISBN: 0-201-48359-9) published by Addison-Wesley. The programs and associated files contained in the distribution were developed by Mark J. Kilgard and are Copyright 1994, 1995, 1996 by Mark J. Kilgard (unless otherwise noted). The programs are not in the public domain, but they are freely distributable without licensing fees. These programs are provided without guarantee or warrantee expressed or implied.

THIS SOURCE CODE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OpenGL (R) is a registered trademark of Silicon Graphics, Inc.

GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February
1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. [This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.] Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below. When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things. To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it. For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they

know their rights. We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library. To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others. Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license. Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs. When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library. We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances. For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use

the Lesser General Public License. In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system. Although the Lesser General Public License is less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library. The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 0.

This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you". A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables. The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".) "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library. Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its

contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a) The modified work must itself be a software library.
 - b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
 - c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
 - d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms

of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library. In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License. 3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices. Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy. This option is useful when you wish to copy part of the code of the Library into a program that is not a library. 4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange. If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code. 5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License. However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a

"work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables. When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law. If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.) Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself. 6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications. You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things: a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified

definitions.) b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with. c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution. d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place. e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as

expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance. 9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it. 10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License. 11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances. It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license

practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS and OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY and REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

OpenNI OpenNI 2.0

Main Page

Namespaces

Classes

Files

OpenNI Overview

Release Notes

OpenNI 2.2.0 Build 33 November 12 2013

Minimum Requirements: -----

- Operating Systems:
 - Windows XP with SP2 and above, Windows 7, Windows 8, on x86 (32/64 bit)
 - Ubuntu 12.04 (32/64/arm) and above
 - Android 2.3 and above
 - Mac OSX 10.7 and above
- Processors:
 - Pentium 4, 1.4GHz and above
 - AMD Athlon 64/FX 1GHz and above
 - Arm Cortex A8 and above
- Memory: at least 64MB available.
- 250MB free hard disk space.
- Available USB 2.0 high-speed port.
- Development Environment:
 - Microsoft Visual Studio 2008 and 2010. The compiler can be MSVC compiler or an Intel Compiler 11 and above.
 - GCC 4.x
- Some of the sample applications require a graphics card equivalent to: ATI RADEON x1300 or NVIDIA GeForce 7300.

Notes: -----

- On Android, only native support (and samples) is currently provided. Please note that as bionic (Android linker) does not support the rpath option, the samples cannot start as is. To solve

this, do one of the following:

- Copy OpenNI libraries (libOpenNI2.so, libPS1080.so and libOniFile.so) to /system/lib (requires root) - or -
 - run `export LD_LIBRARY_PATH=.:\$LD_LIBRARY_PATH` before starting the native executable
-

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[openni::Array< T >](#)

[openni::CameraSettings](#)

[openni::CoordinateConverter](#)

[openni::Device](#)

[openni::OpenNI::DeviceConnectedListener](#)

[openni::OpenNI::DeviceDisconnectedListener](#)

[openni::DeviceInfo](#)

[openni::OpenNI::DeviceStateChangedListener](#)

[openni::VideoStream::FrameAllocator](#)

[openni::VideoStream::NewFrameListener](#)

[openni::OpenNI](#)

[openni::PlaybackControl](#)

[openni::Recorder](#)

[openni::RGB888Pixel](#)

[openni::SensorInfo](#)

[openni::Version](#)

[openni::VideoFrameRef](#)

[openni::VideoMode](#)

[openni::VideoStream](#)

[openni::YUV422DoublePixel](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)[openni](#) › [Array](#) ›

Public Member Functions

openni::Array< T >

Class Template Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

`Array ()`
`Array (const T *data, int count)`
`~Array ()`
int `getSize () const`
const T & `operator[] (int index) const`

Detailed Description

```
template<class T>
class openni::Array< T >
```

Provides a simple array class used throughout the API. Wraps a primitive array of objects, holding the elements and their count.

Constructor & Destructor Documentation

```
template<class T>
openni::Array< T >::Array( ) [inline]
```

Default constructor. Creates an empty **Array** and sets the element count to zero.

```
template<class T>
openni::Array< T >::Array( const T * data,
                           int           count
                           )           [inline]
```

Constructor. Creates new **Array** from an existing primitive array of known size.

Template Parameters:

[in] **T** Object type this **Array** will contain.

Parameters:

[in] **data** Pointer to a primitive array of objects of type **T**.
[in] **count** Number of elements in the primitive array pointed to by **data**.

```
template<class T>
openni::Array< T >::~Array( ) [inline]
```

Destructor. Destroys the **Array** object.

Member Function Documentation

```
template<class T>
int openni::Array< T >::getSize( ) const [inline]
```

Getter function for the **Array** size.

Returns:

Current number of elements in the **Array**.

```
template<class T>
const T& openni::Array< T >::operator[] ( int index ) const [inline]
```

Implements the array indexing operator for the **Array** class.

The documentation for this class was generated from the following file:

- **OpenNI.h**



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [CameraSettings](#) ›

[Public Member Functions](#) | [Friends](#)

openni::CameraSettings Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

```
bool getAutoExposureEnabled () const  
bool getAutoWhiteBalanceEnabled () const  
int getExposure ()  
int getGain ()  
bool isValid () const  
Status setAutoExposureEnabled (bool enabled)  
Status setAutoWhiteBalanceEnabled (bool enabled)  
Status setExposure (int exposure)  
Status setGain (int gain)
```

Friends

```
class VideoStream
```

Member Function Documentation

bool openni::CameraSettings::getAutoExposureEnabled () const

bool openni::CameraSettings::getAutoWhiteBalanceEnabled () const

int openni::CameraSettings::getExposure () [inline]

int openni::CameraSettings::getGain () [inline]

bool openni::CameraSettings::isValid () const [inline]

Status openni::CameraSettings::setAutoExposureEnabled (bool enable)

Status openni::CameraSettings::setAutoWhiteBalanceEnabled (bool enable)

Status openni::CameraSettings::setExposure (int exposure) [inline]

Status openni::CameraSettings::setGain (int gain) [inline]

Friends And Related Function Documentation

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [CoordinateConverter](#) ›

Static Public Member Functions

openni::CoordinateConverter Class Reference

#include <[OpenNI.h](#)>

List of all members.

Static Public Member Functions

static Status	convertDepthToColor (const VideoStream &depthStream, const VideoStream &colorStream, int depthX, int depthY, DepthPixel depthZ, int *pColorX, int *pColorY)
static Status	convertDepthToWorld (const VideoStream &depthStream, int depthX, int depthY, DepthPixel depthZ, float *pWorldX, float *pWorldY, float *pWorldZ)
static Status	convertDepthToWorld (const VideoStream &depthStream, float depthX, float depthY, float depthZ, float *pWorldX, float *pWorldY, float *pWorldZ)
static Status	convertWorldToDepth (const VideoStream &depthStream, float worldX, float worldY, float worldZ, int *pDepthX, int *pDepthY, DepthPixel *pDepthZ)
static Status	convertWorldToDepth (const VideoStream &depthStream, float worldX, float worldY, float worldZ, float *pDepthX, float *pDepthY, float *pDepthZ)

Detailed Description

The [CoordinateConverter](#) class converts points between the different coordinate systems.

Depth and World coordinate systems

[OpenNI](#) applications commonly use two different coordinate systems to represent depth. These two systems are referred to as Depth and World representation.

Depth coordinates are the native data representation. In this system, the frame is a map (two dimensional array), and each pixel is assigned a depth value. This depth value represents the distance between the camera plane and whatever object is in the given pixel. The X and Y coordinates are simply the location in the map, where the origin is the top-left corner of the field of view.

World coordinates superimpose a more familiar 3D Cartesian coordinate system on the world, with the camera lens at the origin. In this system, every point is specified by 3 points -- x, y and z. The x axis of this system is along a line that passes through the infrared projector and CMOS imager of the camera. The y axis is parallel to the front face of the camera, and perpendicular to the x axis (it will also be perpendicular to the ground if the camera is upright and level). The z axis runs into the scene, perpendicular to both the x and y axis. From the perspective of the camera, an object moving from left to right is moving along the increasing x axis. An object moving up is moving along the increasing y axis, and an object moving away from the camera is moving along the increasing z axis.

Mathematically, the Depth coordinate system is the projection of the scene on the CMOS. If the sensor's angular field of view and resolution are known, then an angular size can be calculated for each pixel. This is how the conversion algorithms work. The dependence of this calculation on FoV and resolution is the reason that a [VideoStream](#) pointer must be provided to these functions. The [VideoStream](#) pointer

is used to determine parameters for the specific points to be converted.

Since Depth coordinates are a projective, the apparent size of objects in depth coordinates (measured in pixels) will increase as an object moves closer to the sensor. The size of objects in the World coordinate system is independent of distance from the sensor.

Note that converting from Depth to World coordinates is relatively expensive computationally. It is generally not practical to convert the entire raw depth map to World coordinates. A better approach is to have your computer vision algorithm work in Depth coordinates for as long as possible, and only converting a few specific points to World coordinates right before output.

Note that when converting from Depth to World or vice versa, the Z value remains the same.

Member Function Documentation

static Status openni::CoordinateConverter::convertDepthToColor (

)

For a given depth point, provides the coordinates of the corresponding color value. Useful for superimposing the depth and color images. This operation is the same as turning on registration, but is performed on a single pixel rather than the whole image.

Parameters:

[in] **depthStream** Reference to a [openni::VideoStream](#) that produced the depth value

[in] **colorStream** Reference to a [openni::VideoStream](#) that we want to find the appropriate color pixel in

[in] **depthX** X value of the depth point, given in Depth coordinates and measured in pixels

[in] **depthY** Y value of the depth point, given in Depth coordinates and measured in pixels

[in] **depthZ** Z(depth) value of the depth point, given in the [PixelFormat](#) of depthStream

[out] **pColorX** The X coordinate of the color pixel that overlaps the given depth pixel, measured in pixels

[out]

pColorY	The Y coordinate of the color pixel that overlaps the given depth pixel, measured in pixels
----------------	---

static Status openni::CoordinateConverter::convertDepthToWorld (

)

Converts a single point from the Depth coordinate system to the World coordinate system.

Parameters:

[in] **depthStream** Reference to an openi::VideoStream that will be used to determine the format of the Depth coordinates

[in] **depthX** The X coordinate of the point to be converted, measured in pixels with 0 at the far left of the image

[in] **depthY** The Y coordinate of the point to be converted, measured in pixels with 0 at the top of the image

[in] **depthZ** the Z(depth) coordinate of the point to be converted, measured in the **PixelFormat** of depthStream

[out] **pWorldX** Pointer to a place to store the X coordinate of the output value, measured in millimeters in World coordinates

[out] **pWorldY** Pointer to a place to store the Y

	coordinate of the output value, measured in millimeters in World coordinates
[out] pWorldZ	Pointer to a place to store the Z coordinate of the output value, measured in millimeters in World coordinates

static Status openni::CoordinateConverter::convertDepthToWorld (

)

Converts a single point from a floating point representation of the Depth coordinate system to the World coordinate system.

Parameters:

[in] depthStream	Reference to an openi::VideoStream that will be used to determine the format of the Depth coordinates
[in] depthX	The X coordinate of the point to be converted, measured in pixels with 0.0 at the far left of the image
[in] depthY	The Y coordinate of the point to be converted, measured in pixels with 0.0 at the top of the image
[in] depthZ	Z(depth) coordinate of the point to be converted, measured in the PixelFormat of depthStream
[out] pWorldX	Pointer to a place to store the X

	coordinate of the output value, measured in millimeters in World coordinates
[out] pWorldY	Pointer to a place to store the Y coordinate of the output value, measured in millimeters in World coordinates
[out] pWorldZ	Pointer to a place to store the Z coordinate of the output value, measured in millimeters in World coordinates

static Status openni::CoordinateConverter::convertWorldToDepth (

)

Converts a single point from the World coordinate system to the Depth coordinate system.

Parameters:

[in] depthStream	Reference to an openni::VideoStream that will be used to determine the format of the Depth coordinates
[in] worldX	The X coordinate of the point to be converted, measured in millimeters in World coordinates
[in] worldY	The Y coordinate of the point to be converted, measured in millimeters in World coordinates

[in] worldZ	The Z coordinate of the point to be converted, measured in millimeters in World coordinates
[out] pDepthX	Pointer to a place to store the X coordinate of the output value, measured in pixels with 0 at far left of image
[out] pDepthY	Pointer to a place to store the Y coordinate of the output value, measured in pixels with 0 at top of image
[out] pDepthZ	Pointer to a place to store the Z(depth) coordinate of the output value, measured in the PixelFormat of depthStream

static Status openni::CoordinateConverter::convertWorldToDepth (

Converts a single point from the World coordinate system to a floating point representation of the Depth coordinate system

Parameters:

[in] depthStream	Reference to an openni::VideoStream that will be used to determine the format of the Depth coordinates
[in] worldX	The X coordinate of the point to be converted, measured in millimeters in

	World coordinates
[in] worldY	The Y coordinate of the point to be converted, measured in millimeters in World coordinates
[in] worldZ	The Z coordinate of the point to be converted, measured in millimeters in World coordinates
[out] pDepthX	Pointer to a place to store the X coordinate of the output value, measured in pixels with 0.0 at far left of the image
[out] pDepthY	Pointer to a place to store the Y coordinate of the output value, measured in pixels with 0.0 at the top of the image
[out] pDepthZ	Pointer to a place to store the Z(depth) coordinate of the output value, measured in millimeters with 0.0 at the camera lens

The documentation for this class was generated from the following file:

- [OpenNI.h](#)



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)[openni](#) › [Device](#) ›

Public Member Functions

openni::Device Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

	Device ()
	Device (OniDeviceHandle handle)
	~Device ()
void	close ()
bool	getDepthColorSyncEnabled ()
const DeviceInfo &	getDeviceInfo () const
ImageRegistrationMode	getImageRegistrationMode () const
PlaybackControl *	getPlaybackControl ()
Status	getProperty (int propertyId, void *data, int *dataSize) const
template<class T >	
	Status getProperty (int propertyId, T *value) const
const SensorInfo *	getSensorInfo (SensorType sensorType)
bool	hasSensor (SensorType sensorType)
Status	invoke (int commandId, void *data, int dataSize)
template<class T >	
	Status invoke (int propertyId, T &value)
bool	isCommandSupported (int commandId) const
bool	isFile () const
bool	isImageRegistrationModeSupported (ImageRegistrationMode mode) const
bool	isPropertySupported (int propertyId) const
bool	isValid () const
Status	open (const char *uri)
Status	setDepthColorSyncEnabled (bool isEnabled)
Status	setImageRegistrationMode (ImageRegistrationMode mode)

Status **setProperty** (int propertyId, const void
*data, int dataSize)

template<class T >

Status **setProperty** (int propertyId, const T
&value)

Detailed Description

The **Device** object abstracts a specific device; either a single hardware device, or a file device holding a recording from a hardware device. It offers the ability to connect to the device, and obtain information about its configuration and the data streams it can offer.

It provides the means to query and change all configuration parameters that apply to the device as a whole. This includes enabling depth/color image registration and frame synchronization.

Devices are used when creating and initializing **VideoStreams** -- you will need a valid pointer to a **Device** in order to use the **VideoStream.create()** function. This, along with configuration, is the primary use of this class for application developers.

Before devices can be created, **OpenNI::initialize()** must have been run to make the device drivers on the system available to the API.

Constructor & Destructor Documentation

`openni::Device::Device() [inline]`

Default constructor. Creates a new empty **Device** object. This object will be invalid until it is initialized by calling its **open()** function.

`openni::Device::Device(OniDeviceHandle handle) [inline, explicit]`

Handle constructor. Creates a **Device** object based on the given initialized handle. This object will not destroy the underlying handle when **close()** or destructor is called

`openni::Device::~Device() [inline]`

The destructor calls the **close()** function, but it is considered a best practice for applications to call **close()** manually on any **Device** that they run **open()** on.

Member Function Documentation

void openni::Device::close () [inline]

Closes the device. This properly closes any files or shuts down hardware, as appropriate. This function is currently called by the destructor if not called manually by application code, but it is considered a best practice to manually close any device that was opened.

bool openni::Device::getDepthColorSyncEnabled () [inline]

const DeviceInfo& openni::Device::getDeviceInfo () const [inline]

Provides information about this device in the form of a [DeviceInfo](#) object. This object can be used to access the URI of the device, as well as various USB descriptor strings that might be useful to an application.

Note that valid device info will not be available if this device has not yet been opened. If you are trying to obtain a URI to open a device, use [OpenNI::enumerateDevices\(\)](#) instead.

Returns:

[DeviceInfo](#) object for this [Device](#)

ImageRegistrationMode openni::Device::getImageRegistrationMod

Gets the current image registration mode of this device. Image registration is used to properly superimpose two images from cameras located at different points in space. Please see the OpenNi 2.0 Programmer's Guide for more information about registration.

Returns:

Current image registration mode. See [ImageRegistrationMode](#) for possible return values.

PlaybackControl* `openni::Device::getPlaybackControl ()` [inline]

Gets an object through which playback of a file device can be controlled.

Returns:

NULL if this device is not a file device.

```
Status openni::Device::getProperty ( int      propertyId,
                                         void *   data,
                                         int *    dataSize
                                       )      const [inline]
```

Get the value of a general property of the device. There are convenience functions for all the commonly used properties, such as image registration and frame synchronization. It is expected for this reason that this function will rarely be directly used by applications.

Parameters:

- [in] **propertyId** Numerical ID of the property you would like to check.
- [out] **data** Place to store the value of the property.
- [in, out] **dataSize** IN: Size of the buffer passed in the data argument. OUT: the actual written size.

Returns:

Status code indicating results of this operation.

```
Status openni::Device::getProperty ( int propertyId,
                                         T * value
                                     ) const [inline]
```

Checks a property that provides an arbitrary data type as its output. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Template Parameters:

[in] T Data type of the value to be read.

Parameters:

[in] **propertyId** The numerical ID of the property to be read.
[in, out] **value** Pointer to a place to store the value read from the property.

Returns:

Status code indicating success or failure of this operation.

```
const SensorInfo* openni::Device::getSensorInfo ( SensorType sensorType ) [inline]
```

Get the **SensorInfo** for a specific sensor type on this device. The **SensorInfo** is useful primarily for determining which video modes are supported by the sensor.

Parameters:

[in] **sensorType** of sensor to get information about.

Returns:

SensorInfo object corresponding to the sensor type specified, or NULL if such a sensor is not available from this device.

```
bool openni::Device::hasSensor ( SensorType sensorType ) [inline]
```

This function checks to see if one of the specific sensor types defined in **SensorType** is available on this device. This allows an application to, for example, query for the presence of a depth sensor, or color sensor.

Parameters:

[in] **sensorType** of sensor to query for

Returns:

true if the **Device** supports the sensor queried, false otherwise.

```
Status openni::Device::invoke ( int      commandId,
                                void *   data,
                                int      dataSize
                           )      [inline]
```

Invokes a command that takes an arbitrary data type as its input. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Parameters:

[in] **commandId** Numerical code of the property to be invoked.
[in] **data** Data to be passed to the property.
[in] **dataSize** size of the buffer passed in data.

Returns:

Status code indicating success or failure of this operation.

```
template<class T >
Status openni::Device::invoke ( int  propertyId,
                               T & value
                           )      [inline]
```

Invokes a command that takes an arbitrary data type as its input. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Template Parameters:

[in] T Type of data to be passed to the property.

Parameters:

[in] **propertyId** Numerical code of the property to be invoked.

[in] **value** Data to be passed to the property.

Returns:

Status code indicating success or failure of this operation.

bool openni::Device::isCommandSupported (int commandId) const

Checks if a specific command is supported by the device.

Parameters:

[in] **commandId** Command to be checked.

Returns:

true if the command is supported, false otherwise.

bool openni::Device::isFile () const [inline]

Checks whether this device is a file device (i.e. a recording).

Returns:

true if this is a file device, false otherwise.

bool openni::Device::isImageRegistrationModeSupported (ImageR

Checks to see if this device can support registration of color video and depth video. Image registration is used to properly superimpose two images from cameras located at different points in space. Please see the OpenNi 2.0 Programmer's Guide for more information about registration.

Returns:

true if image registration is supported by this device, false otherwise.

bool openni::Device::isPropertySupported (int `propertyId`) const

Checks if a specific property is supported by the device.

Parameters:

[in] `propertyId` Property to be checked.

Returns:

true if the property is supported, false otherwise.

bool openni::Device::isValid () const [inline]

Checks whether this `Device` object is currently connected to an actual file or hardware device.

Returns:

true if the `Device` is connected, false otherwise.

Status openni::Device::open (const char * `uri`) [inline]

Opens a device. This can either open a device chosen arbitrarily from all devices on the system, or open a specific device selected by passing this function the device URI.

To open any device, simply pass the constant **ANY_DEVICE** to this function. If multiple devices are connected to the system, then one of them will be opened. This procedure is most useful when it is known that exactly one device is (or can be) connected to the system. In that case, requesting a list of all devices and iterating through it would be a waste of effort.

If multiple devices are (or may be) connected to a system, then a URI will be required to select a specific device to open. There are two ways to obtain a URI: from a DeviceConnected event, or by calling **OpenNI::enumerateDevices()**.

In the case of a DeviceConnected event, the OpenNI::Listener will be provided with a **DeviceInfo** object as an argument to its **onDeviceConnected()** function. The **DeviceInfo.getUri()** function can then be used to obtain the URI.

If the application is not using event handlers, then it can also call the static function **OpenNI::enumerateDevices()**. This will return an array of **DeviceInfo** objects, one for each device currently available to the system. The application can then iterate through this list and select the desired device. The URI is again obtained via the **DeviceInfo::getUri()** function.

Standard codes of type Status are returned indicating whether opening was successful.

Parameters:

[in] **uri** String containing the URI of the device to be opened, or **ANY_DEVICE**.

Returns:

Status code with the outcome of the open operation.

Remarks:

For opening a recording file, pass the file path as a uri.

Status `openni::Device::setDepthColorSyncEnabled` (`bool isEnabled`)

Used to turn the depth/color frame synchronization feature on and off. When frame synchronization is enabled, the device will deliver depth and image frames that are separated in time by some maximum value. When disabled, the phase difference between depth and image frame generation cannot be guaranteed.

Parameters:

[in] `isEnabled` Set to TRUE to enable synchronization, FALSE to disable it

Returns:

Status code indicating success or failure of this operation

Status `openni::Device::setImageRegistrationMode` (`ImageRegistrationMode mode`)

Sets the image registration on this device. Image registration is used to properly superimpose two images from cameras located at different points in space. Please see the OpenNi 2.0 Programmer's Guide for more information about registration.

See [ImageRegistrationMode](#) for a list of valid settings to pass to this function.

It is a good practice to first check if the mode is supported by calling [isImageRegistrationModeSupported\(\)](#).

Parameters:

[in] `mode` Desired new value for the image registration mode.

Returns:

Status code for the operation.

```
Status openni::Device::setProperty ( int          propertyId,
                                      const void * data,
                                      int          dataSize
                                    )          [inline]
```

Sets the value of a general property of the device. There are convenience functions for all the commonly used properties, such as image registration and frame synchronization. It is expected for this reason that this function will rarely be directly used by applications.

Parameters:

- [in] **propertyId** The numerical ID of the property to be set.
- [in] **data** Place to store the data to be written to the property.
- [in] **dataSize** Size of the data to be written to the property.

Returns:

Status code indicating results of this operation.

template<class T >

```
Status openni::Device::setProperty ( int          propertyId,
                                      const T & value
                                    )          [inline]
```

Sets a property that takes an arbitrary data type as its input. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Template Parameters:

T Type of data to be passed to the property.

Parameters:

- [in] **propertyId** The numerical ID of the property to be set.
- [in] **value** Place to store the data to be written to the

property.

Returns:

Status code indicating success or failure of this operation.

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [OpenNI](#) › [DeviceConnectedListener](#) ›

[Public Member Functions](#) | [Friends](#)

openni::OpenNI::DeviceConnectedListener Class Reference

#include <[OpenNI.h](#)>

List of all members.

Public Member Functions

```
DeviceConnectedListener ()  
virtual ~DeviceConnectedListener ()  
virtual void onDeviceConnected (const DeviceInfo *)=0
```

Friends

```
class OpenNI
```

Detailed Description

The [**OpenNI::DeviceConnectedListener**](#) class provides a means of registering for, and responding to when a device is connected.

onDeviceConnected is called whenever a new device is connected to the system (ie this event would be triggered when a new sensor is manually plugged into the host system running the application)

To use this class, you should write a new class that inherits from it, and override the onDeviceConnected method. Once you instantiate your class, use the [**OpenNI::addDeviceConnectedListener\(\)**](#) function to add your listener object to OpenNI's list of listeners. Your handler function will then be called whenever the event occurs. A [**OpenNI::removeDeviceConnectedListener\(\)**](#) function is also provided, if you want to have your class stop listening to these events for any reason.

Constructor & Destructor Documentation

openni::OpenNI::DeviceConnectedListener::DeviceConnectedListener

virtual openni::OpenNI::DeviceConnectedListener::~DeviceConnectedListener

Member Function Documentation

virtual void openni::OpenNI::DeviceConnectedListener::onDeviceC

Callback function for the onDeviceConnected event. This function will be called whenever this event occurs. When this happens, a pointer to the **DeviceInfo** object for the newly connected device will be supplied. Note that once a device is removed, if it was opened by a **Device** object, that object can no longer be used to access the device, even if it was reconnected. Once a device was reconnected, **Device::open()** should be called again in order to use this device.

If you wish to open the new device as it is connected, simply query the provided **DeviceInfo** object to obtain the URI of the device, and pass this URI to the **Device.Open()** function.

Friends And Related Function Documentation

friend class OpenNI [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [OpenNI](#) › [DeviceDisconnectedListener](#) ›

[Public Member Functions](#) | [Friends](#)

openni::OpenNI::DeviceDisconnectedListener Class Reference

#include <[OpenNI.h](#)>

List of all members.

Public Member Functions

```
DeviceDisconnectedListener ()  
virtual ~DeviceDisconnectedListener ()  
virtual void onDeviceDisconnected (const DeviceInfo *)=0
```

Friends

```
class OpenNI
```

Detailed Description

The [**OpenNI::DeviceDisconnectedListener**](#) class provides a means of registering for, and responding to when a device is disconnected.

onDeviceDisconnected is called when a device is removed from the system. Note that once a device is removed, if it was opened by a [**Device**](#) object, that object can no longer be used to access the device, even if it was reconnected. Once a device was reconnected, [**Device::open\(\)**](#) should be called again in order to use this device.

To use this class, you should write a new class that inherits from it, and override the onDeviceDisconnected method. Once you instantiate your class, use the [**OpenNI::addDeviceDisconnectedListener\(\)**](#) function to add your listener object to OpenNI's list of listeners. Your handler function will then be called whenever the event occurs. A [**OpenNI::removeDeviceDisconnectedListener\(\)**](#) function is also provided, if you want to have your class stop listening to these events for any reason.

Constructor & Destructor Documentation

openni::OpenNI::DeviceDisconnectedListener::DeviceDisconnecte

virtual openni::OpenNI::DeviceDisconnectedListener::~DeviceDisc

Member Function Documentation

virtual void openni::OpenNI::DeviceDisconnectedListener::onDeviceDisconne

Callback function for the onDeviceDisconnected event. This function will be called whenever this event occurs. When this happens, a pointer to the [DeviceInfo](#) object for the newly disconnected device will be supplied. Note that once a device is removed, if it was opened by a [Device](#) object, that object can no longer be used to access the device, even if it was reconnected. Once a device was reconnected, [Device::open\(\)](#) should be called again in order to use this device.

Friends And Related Function Documentation

friend class OpenNI [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [DeviceInfo](#) ›

[Public Member Functions](#) | [Friends](#)

openni::DeviceInfo

Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

```
const char * getName () const
const char * getUri () const
    uint16_t getUsbProductId () const
    uint16_t getUsbVendorId () const
const char * getVendor () const
```

Friends

```
class Device
```

```
class OpenNI
```

Detailed Description

The [DeviceInfo](#) class encapsulates info related to a specific device.

Applications will generally obtain objects of this type via calls to [OpenNI::enumerateDevices\(\)](#) or [openni::Device::getDeviceInfo\(\)](#), and then use the various accessor functions to obtain specific information on that device.

There should be no reason for application code to instantiate this object directly.

Member Function Documentation

const char* openni::DeviceInfo::getName() const [inline]

Returns the device name for this device.

const char* openni::DeviceInfo::getUri() const [inline]

Returns the device URI. URI can be used by [Device::open](#) to open a specific device. The URI string format is determined by the driver.

uint16_t openni::DeviceInfo::getUsbProductId() const [inline]

Returns the USB PID code for this device.

uint16_t openni::DeviceInfo::getUsbVendorId() const [inline]

Returns the USB VID code for this device.

const char* openni::DeviceInfo::getVendor() const [inline]

Returns the vendor name for this device.

Friends And Related Function Documentation

friend class Device [friend]

friend class OpenNI [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [OpenNI](#) › [DeviceStateChangedListener](#) ›

[Public Member Functions](#) | [Friends](#)

openni::OpenNI::DeviceStateChangedListener Class Reference

#include <[OpenNI.h](#)>

List of all members.

Public Member Functions

```
DeviceStateChangedListener ()  
virtual ~DeviceStateChangedListener ()  
virtual void onDeviceStateChanged (const DeviceInfo *,  
DeviceState)=0
```

Friends

```
class OpenNI
```

Detailed Description

The [**OpenNI::DeviceStateChangedListener**](#) class provides a means of registering for, and responding to when a device's state is changed.

onDeviceStateChanged is triggered whenever the state of a connected device is changed.

To use this class, you should write a new class that inherits from it, and override the onDeviceStateChanged method. Once you instantiate your class, use the [**OpenNI::addDeviceStateChangedListener\(\)**](#) function to add your listener object to OpenNI's list of listeners. Your handler function will then be called whenever the event occurs. A [**OpenNI::removeDeviceStateChangedListener\(\)**](#) function is also provided, if you want to have your class stop listening to these events for any reason.

Constructor & Destructor Documentation

openni::OpenNI::DeviceStateChangedListener::DeviceStateChangedListener

virtual openni::OpenNI::DeviceStateChangedListener::~DeviceStateChangedListener

Member Function Documentation

virtual void openni::OpenNI::DeviceStateChangedListener::onDevi

Callback function for the onDeviceStateChanged event. This function will be called whenever this event occurs. When this happens, a pointer to a [DeviceInfo](#) object for the affected device will be supplied, as well as the new DeviceState value of that device.

Friends And Related Function Documentation

friend class OpenNI [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)[openni](#)[VideoStream](#)[FrameAllocator](#)[Public Member Functions](#) | [Friends](#)

openni::VideoStream::FrameAllocator Class Reference

#include <[OpenNI.h](#)>

List of all members.

Public Member Functions

```
virtual ~FrameAllocator ()  
virtual void * allocateFrameBuffer (int size)=0  
virtual void freeFrameBuffer (void *data)=0
```

Friends

```
class VideoStream
```

Constructor & Destructor Documentation

virtual openni::VideoStream::FrameAllocator::~FrameAllocator()

Member Function Documentation

virtual void* openni::VideoStream::FrameAllocator::allocateFrame()

virtual void openni::VideoStream::FrameAllocator::freeFrameBuffer()

Friends And Related Function Documentation

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

openni

VideoStream

NewFrameListener

[Public Member Functions](#) | [Friends](#)

openni::VideoStream::NewFrameListener Class Reference

#include <OpenNI.h>

List of all members.

Public Member Functions

```
NewFrameListener ()  
virtual ~NewFrameListener ()  
virtual void onNewFrame (VideoStream &)=0
```

Friends

```
class VideoStream
```

Detailed Description

The [**VideoStream::NewFrameListener**](#) class is provided to allow the implementation of event driven frame reading. To use it, create a class that inherits from it and implement override the [**onNewFrame\(\)**](#) method. Then, register your created class with an active [**VideoStream**](#) using the [**VideoStream::addNewFrameListener\(\)**](#) function. Once this is done, the event handler function you implemented will be called whenever a new frame becomes available. You may call [**VideoStream::readFrame\(\)**](#) from within the event handler.

Constructor & Destructor Documentation

openni::VideoStream::NewFrameListener::NewFrameListener() [i]

Default constructor.

virtual openni::VideoStream::NewFrameListener::~NewFrameListe

Member Function Documentation

virtual void openni::VideoStream::NewFrameListener::onNewFram

Derived classes should implement this function to handle new frames.

Friends And Related Function Documentation

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [OpenNI](#) ›

openni::OpenNI Class Reference

[Classes](#) |
[Static Public Member Functions](#)

```
#include <OpenNI.h>
```

List of all members.

Classes

```
class DeviceConnectedListener  
class DeviceDisconnectedListener  
class DeviceStateChangedListener
```

Static Public Member Functions

static Status	addDeviceConnectedListener (DeviceConnectedListener *pListener)
static Status	addDeviceDisconnectedListener (DeviceDisconnectedListener *pListener)
static Status	addDeviceStateChangedListener (DeviceStateChangedListener *pListener)
static void	enumerateDevices (Array< DeviceInfo > *deviceInfoList)
static const char *	getExtendedError ()
static Status	getLogFileName (char *strFileName, int nBufferSize)
static Version	getVersion ()
static Status	initialize ()
static void	removeDeviceConnectedListener (DeviceConnectedListener *pListener)
static void	removeDeviceDisconnectedListener (DeviceDisconnectedListener *pListener)
static void	removeDeviceStateChangedListener (DeviceStateChangedListener *pListener)
static Status	setLogAndroidOutput (bool bAndroidOutput)
static Status	setLogConsoleOutput (bool bConsoleOutput)
static Status	setLogFileOutput (bool bFileOutput)
static Status	setLogMinSeverity (int nMinSeverity)
static Status	setLogOutputFolder (const char *strLogOutputFolder)
static void	shutdown ()
static Status	waitForAnyStream (VideoStream **pStreams, int streamCount, int *pReadyStreamIndex, int timeout= TIMEOUT_FOREVER)

Detailed Description

The **OpenNI** class is a static entry point to the library. It is used by every **OpenNI** 2.0 application to initialize the SDK and drivers to enable creation of valid device objects.

It also defines a listener class and events that enable for event driven notification of device connection, device disconnection, and device configuration changes.

In addition, it gives access to SDK version information and provides a function that allows you to wait for data to become available on any one of a list of streams (as opposed to waiting for data on one specific stream with functions provided by the **VideoStream** class)

Member Function Documentation

static Status openni::OpenNI::addDeviceConnectedListener (DeviceConnectedListener* pListener)

Add a listener to the list of objects that receive the event when a device is connected. See the [OpenNI::DeviceConnectedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be added to the list

Returns:

Status code indicating success or failure of this operation.

static Status openni::OpenNI::addDeviceDisconnectedListener (DeviceDisconnectedListener* pListener)

Add a listener to the list of objects that receive the event when a device is disconnected. See the [OpenNI::DeviceDisconnectedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be added to the list

Returns:

Status code indicating success or failure of this operation.

static Status openni::OpenNI::addDeviceStateChangedListener (DeviceStateChangedListener* pListener)

Add a listener to the list of objects that receive the event when a device's state changes. See the [OpenNI::DeviceStateChangedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be added to the list

Returns:

Status code indicating success or failure of this operation.

static void openni::OpenNI::enumerateDevices ([Array< DeviceInfo](#) &

Fills up an array of [DeviceInfo](#) objects with devices that are available.

Parameters:

[in, out] **deviceInfoList** An array to be filled with devices.

static const char* openni::OpenNI::getExtendedError () [inline, s

Retrieves the calling thread's last extended error information. The last extended error information is maintained on a per-thread basis.

Multiple threads do not overwrite each other's last extended error information.

The extended error information is cleared on every call to an [OpenNI](#) method, so you should call this method immediately after a call to an [OpenNI](#) method which have failed.

static Status openni::OpenNI::getLogFileName (char * strFileName, int nBufferSize) [inline, sta

Get current log file name

Parameters:

char * strFileName [out] returned file name buffer

int nBufferSize [in] Buffer size

Returns:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static Version openni::OpenNI::getVersion () [inline, static]

Returns the version of [OpenNI](#)

static Status openni::OpenNI::initialize () [inline, static]

Initialize the library. This will load all available drivers, and see which devices are available It is forbidden to call any other method in [OpenNI](#) before calling [initialize\(\)](#).

static void openni::OpenNI::removeDeviceConnectedListener (Dev

Remove a listener from the list of objects that receive the event when a device is connected. See the [OpenNI::DeviceConnectedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be removed from the list

Returns:

Status code indicating the success or failure of this operation.

static void openni::OpenNI::removeDeviceDisconnectedListener (I

Remove a listener from the list of objects that receive the event when a device is disconnected. See the

[OpenNI::DeviceDisconnectedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be removed from the list

Returns:

Status code indicating the success or failure of this operation.

static void openni::OpenNI::removeDeviceStateChangedListener (

Remove a listener from the list of objects that receive the event when a device's state changes. See the [OpenNI::DeviceStateChangedListener](#) class for details on utilizing the events provided by [OpenNI](#).

Parameters:

pListener Pointer to the Listener to be removed from the list

Returns:

Status code indicating the success or failure of this operation.

static Status openni::OpenNI::setLogAndroidOutput (bool bAndro

Configures if log entries will be printed to the Android log.

Parameters:

OniBool bAndroidOutput bAndroidOutput [in] TRUE to print log entries to the Android log, FALSE otherwise.

Return values:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static Status openni::OpenNI::setLogConsoleOutput (bool bConsoleOutput)

Configures if log entries will be printed to console.

Parameters:

const OniBool bConsoleOutput [in] TRUE to print log entries to console, FALSE otherwise.

Return values:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static Status openni::OpenNI::setLogFileOutput (bool bFileOutput)

Configures if log entries will be printed to file.

Parameters:

const OniBool bFileOutput [in] TRUE to print log entries to file, FALSE otherwise.

Return values:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static Status openni::OpenNI::setLogMinSeverity (int nMinSeverity)

Set minimum severity for log produce

Parameters:

const char * strMask [in] Logger name

int nMinSeverity [in] Logger severity

Return values:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static Status openni::OpenNI::setLogOutputFolder (const char * s1)

Change the log output folder

Parameters:

const char * strLogOutputFolder [in] log required folder

Return values:

STATUS_OK Upon successful completion.

STATUS_ERROR Upon any kind of failure.

static void openni::OpenNI::shutdown () [inline, static]

Stop using the library. Unload all drivers, close all streams and devices. Once **shutdown** was called, no other calls to **OpenNI** is allowed.

static Status openni::OpenNI::waitForAnyStream (VideoStream ** pStreams, int streamCount, int timeout)

Wait for a new frame from any of the streams provided. The function blocks until any of the streams has a new frame available, or the timeout has passed.

Parameters:

[in] pStreams An array of streams to wait for.

[in] streamCount The number of streams in pStreams

[out] pReadyStreamIndex	The index of the first stream that has new frame available.
[in] timeout	[Optional] A timeout before returning if no stream has new data. Default value is TIMEOUT_FOREVER .

The documentation for this class was generated from the following file:

- [OpenNI.h](#)



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [PlaybackControl](#) ›

[Public Member Functions](#) | [Friends](#)

openni::PlaybackControl Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

```
~PlaybackControl ()  
int getNumberOfFrames (const VideoStream &stream) const  
bool getRepeatEnabled () const  
float getSpeed () const  
bool isValid () const  
Status seek (const VideoStream &stream, int frameIndex)  
Status setRepeatEnabled (bool repeat)  
Status setSpeed (float speed)
```

Friends

class Device

Detailed Description

The [PlaybackControl](#) class provides access to a series of specific to playing back a recording from a file device.

When playing a stream back from a recording instead of playing from a live device, it is possible to vary playback speed, change the current time location (ie fast forward / rewind / seek), specify whether the playback should be repeated at the end of the recording, and query the total size of the recording.

Since none of these functions make sense in the context of a physical device, they are split out into a separate playback control class. To use, simply create your file device, create a [PlaybackControl](#), and then attach the [PlaybackControl](#) to the file device.

Constructor & Destructor Documentation

openni::PlaybackControl::~PlaybackControl() [inline]

Deconstructor. Destroys a **PlaybackControl** class. The deconstructor presently detaches from its recording automatically, but it is considered a best practice for applications to manually detach from any stream that was attached to.

Member Function Documentation

`int openni::PlaybackControl::getNumberOfFrames (const VideoStr`

Provides the a count of frames that this recording contains for a given stream. This is useful both to determine the length of the recording, and to ensure that a valid Frame Index is set when using the [PlaybackControl::seek\(\)](#) function.

Parameters:

[in] **stream** The video stream to count frames for

Returns:

Number of frames in provided [VideoStream](#), or 0 if the stream is not part of the recording

`bool openni::PlaybackControl::getRepeatEnabled () const [inline]`

Gets the current repeat setting of the file device.

Returns:

true if repeat is enabled, false if not enabled.

`float openni::PlaybackControl::getSpeed () const [inline]`

Getter function for the current playback speed of this device.

This value is expressed as a multiple of the speed the original recording was taken at. For example, if the original recording was at 30fps, and playback speed is set to 0.5, then the recording will play at 15fps. If playback speed is set to 2.0, then the recording would playback at 60fps.

In addition, there are two "special" values. A playback speed of 0.0

indicates that the playback should occur as fast as the system is capable of returning frames. This is most useful when testing algorithms on large datasets, as it enables playback to be done at a much higher rate than would otherwise be possible.

A value of -1 indicates that speed is "manual". In this mode, new frames will only become available when an application manually reads them. If used in a polling loop, this setting also enables systems to read and process frames limited only by available processing speeds.

Returns:

Current playback speed of the device, measured as ratio of recording speed.

bool openni::PlaybackControl::isValid () const [inline]

Status openni::PlaybackControl::seek (const VideoStream & stream, int frameIndex) [inline]

Seeks within a **VideoStream** to a given FrameID. Note that when this function is called on one stream, all other streams will also be changed to the corresponding place in the recording. The FrameIDs of different streams may not match, since FrameIDs may differ for streams that are not synchronized, but the recording will set all streams to the same moment in time.

Parameters:

[in] **stream** Stream for which the frameIndex value is valid.

[in] **frameIndex** Frame index to move playback to

Returns:

Status code indicating success or failure of this operation

Status `openni::PlaybackControl::setRepeatEnabled (bool repeat)`

Changes the current repeat mode of the device. If repeat mode is turned on, then the recording will begin playback again at the beginning after the last frame is read. If turned off, no more frames will become available after last frame is read.

Parameters:

[in] `repeat` New value for repeat -- true to enable, false to disable

Returns:

Status code indicating success or failure of this operations.

Status `openni::PlaybackControl::setSpeed (float speed) [inline]`

Setter function for the playback speed of the device. For a full explaination of what this value means

See also:

[PlaybackControl::getSpeed\(\)](#).

Parameters:

[in] `speed` Desired new value of playback speed, as ratio of original recording.

Returns:

Status code indicating success or failure of this operation.

Friends And Related Function Documentation

friend class Device [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [Recorder](#) ›

Public Member Functions

openni::Recorder Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

Recorder ()

~Recorder ()

Status attach (VideoStream &stream, bool allowLossyCompression=false)

Status create (const char *fileName)

void destroy ()

bool isValid () const

Status start ()

void stop ()

Detailed Description

The [Recorder](#) class is used to record streams to an ONI file.

After a recorder is instantiated, it must be initialized with a specific filename where the recording will be stored. The recorder is then attached to one or more streams. Once this is complete, the recorder can be told to start recording. The recorder will store every frame from every stream to the specified file. Later, this file can be used to initialize a file [Device](#), and used to play back the same data that was recorded.

Opening a file device is done by passing its path as the uri to the [Device::open\(\)](#) method.

See also:

[PlaybackControl](#) for options available to play a reordered file.

Constructor & Destructor Documentation

openni::Recorder::Recorder() [inline]

Creates a recorder. The recorder is not valid, i.e. **isValid()** returns false. You must initialize the recorder before use with **create()**.

openni::Recorder::~Recorder() [inline]

Destroys a recorder. This will also stop recording.

Member Function Documentation

Status `openni::Recorder::attach (VideoStream & stream,`
`bool allowLossyCompr,`
`)` `[inline]`

Attaches a stream to the recorder. Note, this won't start recording, you should explicitly start it using `start()` method. As soon as the recording process has been started, no more streams can be attached to the recorder.

Parameters:

`[in] stream` The stream to be recorded.
`[in] allowLossyCompression` [Optional] If this value is true, the recorder might use a lossy compression, which means that when the recording will be played-back, there might be small differences from the original frame. Default value is false.

Status `openni::Recorder::create (const char * fileName) [inline]`

Initializes a recorder. You can initialize the recorder only once. Attempts to intialize more than once will result in an error code being returned.

Initialization assigns the recorder to an output file that will be used for recording. Before use, the `attach()` function must also be used to assign input data to the `Recorder`.

Parameters:

`[in] fileName` The name of a file which will contain the

recording.

Returns:

Status code which indicates success or failure of the operation.

void openni::Recorder::destroy() [inline]

Destroys the recorder object.

bool openni::Recorder::isValid() const [inline]

Verifies if the recorder is valid, i.e. if one can record with this recorder. A recorder object is not valid until the [create\(\)](#) method is called.

Returns:

true if the recorder has been initialized, false otherwise.

Status openni::Recorder::start() [inline]

Starts recording. Once this method is called, the recorder will take all subsequent frames from the attached streams and store them in the file. You may not attach additional streams once recording was started.

void openni::Recorder::stop() [inline]

Stops recording. You may use [start\(\)](#) to resume the recording.

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)[openni](#) › [RGB888Pixel](#) ›

Public Attributes

openni::RGB888Pixel Struct Reference

```
#include <OpenNI.h>
```

List of all members.

Public Attributes

uint8_t **b**

uint8_t **g**

uint8_t **r**

Detailed Description

Holds the value of a single color image pixel in 24-bit RGB format.

Member Data Documentation

uint8_t `openni::RGB888Pixel::b`

uint8_t `openni::RGB888Pixel::g`

uint8_t `openni::RGB888Pixel::r`

The documentation for this struct was generated from the following file:

- **`OpenNI.h`**

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by **[doxygen](#)** 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [SensorInfo](#) ›

[Public Member Functions](#) | [Friends](#)

openni::SensorInfo Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

`SensorType getSensorType () const`
const `Array< VideoMode > & getSupportedVideoModes () const`

Friends

class Device

class VideoStream

Detailed Description

The **SensorInfo** class encapsulates all info related to a specific sensor in a specific device. A **Device** object holds a **SensorInfo** object for each sensor it contains. A **VideoStream** object holds one **SensorInfo** object, describing the sensor used to produce that stream.

A given **SensorInfo** object will contain the type of the sensor (Depth, IR or Color), and a list of all video modes that the sensor can support. Each available video mode will have a single **VideoMode** object that can be queried to get the details of that mode.

SensorInfo objects should be the only source of **VideoMode** objects for the vast majority of application programs.

Application programs will never directly instantiate objects of type **SensorInfo**. In fact, no public constructors are provided. **SensorInfo** objects should be obtained either from a **Device** or **VideoStream**, and in turn be used to provide available video modes for that sensor.

Member Function Documentation

SensorType `openni::SensorInfo::getSensorType() const [inline]`

Provides the sensor type of the sensor this object is associated with.

Returns:

Type of the sensor.

const Array<VideoMode>& `openni::SensorInfo::getSupportedVideoModes() const [inline]`

Provides a list of video modes that this sensor can support. This function is the recommended method to be used by applications to obtain **VideoMode** objects.

Returns:

Reference to an array of **VideoMode** objects, one for each supported video mode.

Friends And Related Function Documentation

friend class Device [friend]

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)[openni](#) › [Version](#) ›

Public Attributes

openni::Version Struct Reference

```
#include <OpenNI.h>
```

List of all members.

Public Attributes

int **build**
int **maintenance**
int **major**
int **minor**

Detailed Description

Holds an [OpenNI](#) version number, which consists of four separate numbers in the format: `major.minor.maintenance.build`. For example: 2.0.0.20.

Member Data Documentation

int openni::Version::build

Build number. Incremented for each new API build. Generally not shown on the installer and download site.

int openni::Version::maintenance

Maintenance build number, incremented for new releases that primarily provide minor bug fixes.

int openni::Version::major

Major version number, incremented for major API restructuring.

int openni::Version::minor

Minor version number, incremented when significant new features added.

The documentation for this struct was generated from the following file:

- [OpenNI.h](#)



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [VideoFrameRef](#) ›

[Public Member Functions](#) | [Friends](#)

openni::VideoFrameRef Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

	<code>VideoFrameRef ()</code>
	<code>VideoFrameRef (const VideoFrameRef &other)</code>
	<code>~VideoFrameRef ()</code>
int	<code>getCropOriginX () const</code>
int	<code>getCropOriginY () const</code>
bool	<code>getCroppingEnabled () const</code>
const void *	<code>getData () const</code>
int	<code>getDataSize () const</code>
int	<code>getFrameIndex () const</code>
int	<code>getHeight () const</code>
<code>SensorType</code>	<code>getSensorType () const</code>
int	<code>getStrideInBytes () const</code>
uint64_t	<code>getTimestamp () const</code>
const <code>VideoMode</code> &	<code>getVideoMode () const</code>
int	<code>getWidth () const</code>
bool	<code>isValid () const</code>
<code>VideoFrameRef &</code>	<code>operator= (const VideoFrameRef &other)</code>
void	<code>release ()</code>

Friends

```
class VideoStream
```

Detailed Description

The **VideoFrameRef** class encapsulates a single video frame - the output of a **VideoStream** at a specific time. The data contained will be a single frame of color, IR, or depth video, along with associated meta data.

An object of type **VideoFrameRef** does not actually hold the data of the frame, but only a reference to it. The reference can be released by destroying the **VideoFrameRef** object, or by calling the **release()** method. The actual data of the frame is freed when the last reference to it is released.

The usual way to obtain **VideoFrameRef** objects is by a call to **VideoStream**::**readFrame()**.

All data references by a **VideoFrameRef** is stored as a primitive array of pixels. Each pixel will be of a type according to the configured pixel format (see **VideoMode**).

Constructor & Destructor Documentation

openni::VideoFrameRef::VideoFrameRef() [inline]

Default constructor. Creates a new empty **VideoFrameRef** object. This object will be invalid until initialized by a call to **VideoStream::readFrame()**.

openni::VideoFrameRef::~VideoFrameRef() [inline]

Destroy this object and release the reference to the frame.

openni::VideoFrameRef::VideoFrameRef(const [VideoFrameRef](#) &

Copy constructor. Creates a new **VideoFrameRef** object. The newly created object will reference the same frame current object references.

Parameters:

[in] other Another **VideoFrameRef** object.

Member Function Documentation

int openni::VideoFrameRef::getCropOriginX() const [inline]

Indicates the X coordinate of the upper left corner of the crop window.

Returns:

Distance of crop origin from left side of image, in pixels.

int openni::VideoFrameRef::getCropOriginY() const [inline]

Indicates the Y coordinate of the upper left corner of the crop window.

Returns:

Distance of crop origin from top of image, in pixels.

bool openni::VideoFrameRef::getCroppingEnabled() const [inlin

Indicates whether cropping was enabled when the frame was produced.

Returns:

true if cropping is enabled, false otherwise

const void* openni::VideoFrameRef::getData() const [inline]

Getter function for the array of data pointed to by this object.

Returns:

Pointer to the actual frame data array. Type of data pointed to can be determined according to the pixel format (can be obtained by calling [getVideoMode\(\)](#)).

int openni::VideoFrameRef::getDataSize () const [inline]

Getter function for the size of the data contained by this object. Useful primarily when allocating buffers.

Returns:

Current size of data pointed to by this object, measured in bytes.

int openni::VideoFrameRef::getFrameIndex () const [inline]

Frames are provided sequential frame ID numbers by the sensor that produced them. If frame synchronization has been enabled for a device via [**Device::setDepthColorSyncEnabled\(\)**](#), then frame numbers for corresponding frames of depth and color are guaranteed to match.

If frame synchronization is not enabled, then there is no guarantee of matching frame indexes between [**VideoStreams**](#). In the latter case, applications should use timestamps instead of frame indexes to align frames in time.

Returns:

Index number for this frame.

int openni::VideoFrameRef::getHeight () const [inline]

Gives the current height of this frame, measured in pixels. If cropping is enabled, this will be the length of the cropping window. If cropping is not enabled, then this will simply be equal to the Y resolution of the [**VideoMode**](#) used to produce this frame.

SensorType openni::VideoFrameRef::getSensorType () const [inl]

Getter function for the sensor type used to produce this frame. Used to determine whether this is an IR, Color or Depth frame. See the **SensorType** enumeration for all possible return values from this function.

Returns:

The type of sensor used to produce this frame.

`int openni::VideoFrameRef::getStrideInBytes() const [inline]`

Gives the length of one row of pixels, measured in bytes. Primarily useful for indexing the array which contains the data.

Returns:

Stride of the array which contains the image for this frame, in bytes

`uint64_t openni::VideoFrameRef::getTimestamp() const [inline]`

Provides a timestamp for the frame. The 'zero' point for this stamp is implementation specific, but all streams from the same device are guaranteed to use the same zero. This value can therefore be used to compute time deltas between frames from the same device, regardless of whether they are from the same stream.

Returns:

Timestamp of frame, measured in microseconds from an arbitrary zero

`const VideoMode& openni::VideoFrameRef::getVideoMode() const`

Returns a reference to the **VideoMode** object assigned to this frame. This object describes the video mode the sensor was configured to

when the frame was produced and can be used to determine the pixel format and resolution of the data. It will also provide the frame rate that the sensor was running at when it recorded this frame.

Returns:

Reference to the [VideoMode](#) assigned to this frame.

int openni::VideoFrameRef::getWidth() const [inline]

Gives the current width of this frame, measured in pixels. If cropping is enabled, this will be the width of the cropping window. If cropping is not enabled, then this will simply be equal to the X resolution of the [VideoMode](#) used to produce this frame.

Returns:

Width of this frame in pixels.

bool openni::VideoFrameRef::isValid() const [inline]

Check if this object references an actual frame.

VideoFrameRef& openni::VideoFrameRef::operator=(const VideoF

Make this [VideoFrameRef](#) object reference the same frame that the other frame references. If this object referenced another frame before calling this method, the previous frame will be released.

Parameters:

[in] **other** Another [VideoFrameRef](#) object.

void openni::VideoFrameRef::release() [inline]

Release the reference to the frame. Once this method is called, the

object becomes invalid, and no method should be called other than the assignment operator, or passing this object to a [**VideoStream::readFrame\(\)**](#) call.

Friends And Related Function Documentation

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

[openni](#) › [VideoMode](#) ›

[Public Member Functions](#) | [Friends](#)

openni::VideoMode

Class Reference

```
#include <OpenNI.h>
```

List of all members.

Public Member Functions

	<code>VideoMode ()</code>
	<code>VideoMode (const VideoMode &other)</code>
int	<code>getFps () const</code>
<code>PixelFormat</code>	<code>getPixelFormat () const</code>
int	<code>getResolutionX () const</code>
int	<code>getResolutionY () const</code>
<code>VideoMode &</code>	<code>operator= (const VideoMode &other)</code>
void	<code>setFps (int fps)</code>
void	<code>setPixelFormat (PixelFormat format)</code>
void	<code>setResolution (int resolutionX, int resolutionY)</code>

Friends

```
class SensorInfo  
class VideoFrameRef  
class VideoStream
```

Detailed Description

Encapsulates a group of settings for a [VideoStream](#). Settings stored include frame rate, resolution, and pixel format.

This class is used as an input for changing the settings of a [VideoStream](#), as well as an output for reporting the current settings of that class. It is also used by [SensorInfo](#) to report available video modes of a stream.

Recommended practice is to use [SensorInfo::getSupportedVideoModes\(\)](#) to obtain a list of valid video modes, and then to use items from that list to pass new settings to [VideoStream](#). This is much less likely to produce an invalid video mode than instantiating and manually changing objects of this class.

Constructor & Destructor Documentation

`openni::VideoMode::VideoMode() [inline]`

Default constructor, creates an empty `VideoMode` object. Application programs should, in most cases, use the copy constructor to copy an existing valid video mode. This is much less error prone than creating and attempting to configure a new `VideoMode` from scratch.

`openni::VideoMode::VideoMode(const VideoMode & other) [inline]`

Copy constructor, creates a new `VideoMode` identical to an existing `VideoMode`.

Parameters:

[in] `other` Existing `VideoMode` to copy.

Member Function Documentation

int openni::VideoMode::getFps() const [inline]

Getter function for the frame rate of this **VideoMode**.

Returns:

Current frame rate, measured in frames per second.

PixelFormat openni::VideoMode::getPixelFormat() const [inline]

Getter function for the pixel format of this **VideoMode**.

Returns:

Current pixel format setting of this **VideoMode**.

int openni::VideoMode::getResolutionX() const [inline]

Getter function for the X resolution of this **VideoMode**.

Returns:

Current horizontal resolution of this **VideoMode**, in pixels.

int openni::VideoMode::getResolutionY() const [inline]

Getter function for the Y resolution of this **VideoMode**.

Returns:

Current vertical resolution of this **VideoMode**, in pixels.

VideoMode& openni::VideoMode::operator= (const VideoMode &

Assignment operator. Sets the pixel format, frame rate, and resolution of this [VideoMode](#) to equal that of a different [VideoMode](#).

Parameters:

[in] **other** Existing [VideoMode](#) to copy settings from.

void openni::VideoMode::setFps (int **fps) [inline]**

Setter function for the frame rate. Application use of this function is not recommended. Instead, use

[SensorInfo::getSupportedVideoModes\(\)](#) to obtain a list of valid video modes.

Parameters:

[in] **fps** Desired new frame rate, measured in frames per second.

void openni::VideoMode::setPixelFormat (PixelFormat **format) [in]**

Setter function for the pixel format of this [VideoMode](#). Application use of this function is not recommended. Instead, use

[SensorInfo::getSupportedVideoModes\(\)](#) to obtain a list of valid video modes.

Parameters:

[in] **format** Desired new pixel format for this [VideoMode](#).

void openni::VideoMode::setResolution (int **resolutionX, int **resolutionY**) [inline]**

Setter function for the resolution of this [VideoMode](#). Application use

of this function is not recommended. Instead, use [**SensorInfo::getSupportedVideoModes\(\)**](#) to obtain a list of valid video modes.

Parameters:

- [in] **resolutionX** Desired new horizontal resolution in pixels.
- [in] **resolutionY** Desired new vertical resolution in pixels.

Friends And Related Function Documentation

friend class SensorInfo [friend]

friend class VideoFrameRef [friend]

friend class VideoStream [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)
-

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

openni

VideoStream

[Classes](#) | [Public Member Functions](#)
| [Friends](#)

openni::VideoStream Class Reference

```
#include <OpenNI.h>
```

List of all members.

Classes

class **FrameAllocator**

class **NewFrameListener**

Public Member Functions

	VideoStream ()
	VideoStream (OniStreamHandle handle)
	~VideoStream ()
Status	addNewFrameListener (NewFrameListener *pListener)
Status	create (const Device &device, SensorType sensorType)
void	destroy ()
CameraSettings *	getCameraSettings ()
bool	getCropping (int *pOriginX, int *pOriginY, int *pWidth, int *pHeight) const
float	getHorizontalFieldOfView () const
int	getMaxPixelValue () const
int	getMinPixelValue () const
bool	getMirroringEnabled () const
Status	getProperty (int propertyId, void *data, int *dataSize) const
template<class T >	
	Status getProperty (int propertyId, T *value) const
const SensorInfo &	getSensorInfo () const
	float getVerticalFieldOfView () const
VideoMode	getVideoMode () const
	Status invoke (int commandId, void *data, int dataSize)
template<class T >	
	Status invoke (int commandId, T &value)
	bool isCommandSupported (int commandId) const
	bool isCroppingSupported () const
	bool isPropertySupported (int propertyId) const
	bool isValid () const
	Status readFrame (VideoFrameRef *pFrame)
	void removeNewFrameListener (NewFrameListener *pListener)

Status	resetCropping ()
Status	setCropping (int originX, int originY, int width, int height)
Status	 setFrameBuffersAllocator (FrameAllocator *pAllocator)
Status	setMirroringEnabled (bool isEnabled)
Status	setProperty (int propertyId, const void *data, int dataSize)
template<class T >	
	Status setProperty (int propertyId, const T &value)
	Status setVideoMode (const VideoMode &videoMode)
	Status start ()
	void stop ()

Friends

class Device

Detailed Description

The **VideoStream** object encapsulates a single video stream from a device. Once created, it is used to start data flow from the device, and to read individual frames of data. This is the central class used to obtain data in **OpenNI**. It provides the ability to manually read data in a polling loop, as well as providing events and a Listener class that can be used to implement event-driven data acquisition.

Aside from the video data frames themselves, the class offers a number of functions used for obtaining information about a **VideoStream**. Field of view, available video modes, and minimum and maximum valid pixel values can all be obtained.

In addition to obtaining data, the **VideoStream** object is used to set all configuration properties that apply to a specific stream (rather than to an entire device). In particular, it is used to control cropping, mirroring, and video modes.

A pointer to a valid, initialized device that provides the desired stream type is required to create a stream.

Several video streams can be created to stream data from the same sensor. This is useful if several components of an application need to read frames separately.

While some device might allow different streams from the same sensor to have different configurations, most devices will have a single configuration for the sensor, shared by all streams.

Constructor & Destructor Documentation

`openni::VideoStream::VideoStream()` [inline]

Default constructor. Creates a new, non-valid `VideoStream` object. The object created will be invalid until its `create()` function is called with a valid `Device`.

`openni::VideoStream::VideoStream(OniStreamHandle handle)` [inl]

Handle constructor. Creates a `VideoStream` object based on the given initialized handle. This object will not destroy the underlying handle when `destroy()` or destructor is called

`openni::VideoStream::~VideoStream()` [inline]

Destructor. The destructor calls the `destroy()` function, but it is considered a best practice for applications to call `destroy()` manually on any `VideoStream` that they run `create()` on.

Member Function Documentation

Status `openni::VideoStream::addNewFrameListener (NewFrameLi:`

Adds a new Listener to receive this `VideoStream` onNewFrame event. See `VideoStream::NewFrameListener` for more information on implementing an event driven frame reading architecture. An instance of a listener can be added to only one source.

Parameters:

[in] `pListener` Pointer to a `VideoStream::NewFrameListener` object (or a derivative) that will respond to this event.

Returns:

Status code indicating success or failure of the operation.

Status `openni::VideoStream::create (const Device & device,` `SensorType sensorType` `)` [inline]

Creates a stream of frames from a specific sensor type of a specific device. You must supply a reference to a `Device` that supplies the sensor type requested. You can use `Device::hasSensor()` to check whether a given sensor is available on your target device before calling `create()`.

Parameters:

[in] `device` A reference to the `Device` you want to create the stream on.
[in] `sensorType` The type of sensor the stream should produce data from.

Returns:

Status code indicating success or failure for this operation.

void openni::VideoStream::destroy() [inline]

Destroy this stream. This function is currently called automatically by the destructor, but it is considered a best practice for applications to manually call this function on any **VideoStream** that they call **create()** for.

CameraSettings* openni::VideoStream::getCameraSettings() [inline]

Gets an object through which several camera settings can be configured.

Returns:

NULL if the stream doesn't support camera settings.

**bool openni::VideoStream::getCropping(int * pOriginX,
int * pOriginY,
int * pWidth,
int * pHeight
) const [inline]**

Obtains the current cropping settings for this stream.

Parameters:

[out] **pOriginX** X coordinate of the upper left corner of the cropping window

[out] **pOriginY** Y coordinate of the upper left corner of the cropping window

[out] **pWidth** Horizontal width of the cropping window, in pixels

[out] pHeight Vertical width of the cropping window, in pixels returns true if cropping is currently enabled, false if it is not.

float openni::VideoStream::getHorizontalFieldOfView() const [in]

Gets the horizontal field of view of frames received from this stream.

Returns:

Horizontal field of view, in radians.

int openni::VideoStream::getMaxPixelValue() const [inline]

Provides the maximum possible value for pixels obtained by this stream. This is most useful for getting the maximum possible value of depth streams.

Returns:

Maximum possible pixel value.

int openni::VideoStream::getMinPixelValue() const [inline]

Provides the smallest possible value for pixels obtains by this **VideoStream**. This is most useful for getting the minimum possible value that will be reported by a depth stream.

Returns:

Minimum possible pixel value that can come from this stream.

bool openni::VideoStream::getMirroringEnabled() const [inline]

Check whether mirroring is currently turned on for this stream.

Returns:

true if mirroring is currently enabled, false otherwise.

```
Status openni::VideoStream::getProperty ( int      propertyId,
                                         void *   data,
                                         int *    dataSize
                                       )      const [inline]
```

General function for obtaining the value of stream specific properties. There are convenience functions available for all commonly used properties, so it is not expected that applications will make direct use of the getProperty function very often.

Parameters:

[in] propertyId	The numerical ID of the property to be queried.
[out] data	Place to store the value of the property.
[in, out] dataSize	IN: Size of the buffer passed in the data argument. OUT: the actual written size.

Returns:

Status code indicating success or failure of this operation.

```
template<class T >
```

```
Status openni::VideoStream::getProperty ( int  propertyId,
                                         T *   value
                                       )      const [inline]
```

Function for getting the value from a property using an arbitrary output type. There are convenience functions available for all commonly used properties, so it is not expected that applications will make direct use of this function very often.

Template Parameters:

[in] T Data type of the value to be read.

Parameters:

[in] **propertyId** The numerical ID of the property to be read.

[in, out] **value** Pointer to a place to store the value read from the property.

Returns:

Status code indicating success or failure of this operation.

const SensorInfo& openni::VideoStream::getSensorInfo() const [

Provides the **SensorInfo** object associated with the sensor that is producing this **VideoStream**. Note that this function will return NULL if the stream has not yet been initialized with the **create()** function.

SensorInfo is useful primarily as a means of learning which video modes are valid for this **VideoStream**.

Returns:

Reference to the **SensorInfo** object associated with the sensor providing this stream.

float openni::VideoStream::getVerticalFieldOfView() const [inline]

Gets the vertical field of view of frames received from this stream.

Returns:

Vertical field of view, in radians.

VideoMode openni::VideoStream::getVideoMode() const [inline]

Get the current video mode information for this video stream. This includes its resolution, fps and stream format.

Returns:

Current video mode information for this video stream.

```
Status openni::VideoStream::invoke ( int      commandId,  
                                     void *   data,  
                                     int      dataSize  
                               )      [inline]
```

Invokes a command that takes an arbitrary data type as its input. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Parameters:

- [in] **commandId** Numerical code of the property to be invoked.
- [in] **data** Data to be passed to the property.
- [in] **dataSize** size of the buffer passed in data.

Returns:

Status code indicating success or failure of this operation.

```
template<class T >  
Status openni::VideoStream::invoke ( int  commandId,  
                                    T &  value  
                              )  [inline]
```

Invokes a command that takes an arbitrary data type as its input. It is not expected that application code will need this function frequently, as all commonly used properties have higher level functions provided.

Template Parameters:

[in] T Type of data to be passed to the property.

Parameters:

[in] **commandId** Numerical code of the property to be invoked.

[in] **value** Data to be passed to the property.

Returns:

Status code indicating success or failure of this operation.

bool openni::VideoStream::isCommandSupported (int commandId)

Checks if a specific command is supported by the video stream.

Parameters:

[in] **commandId** Command to be checked.

Returns:

true if the command is supported, false otherwise.

bool openni::VideoStream::isCroppingSupported () const [inline]

Checks whether this stream supports cropping.

Returns:

true if the stream supports cropping, false if it does not.

bool openni::VideoStream::isPropertySupported (int propertyId) c

Checks if a specific property is supported by the video stream.

Parameters:

[in] **propertyId** Property to be checked.

Returns:

true if the property is supported, false otherwise.

bool openni::VideoStream::isValid () const [inline]

Checks to see if this object has been properly initialized and currently points to a valid stream.

Returns:

true if this object has been previously initialized, false otherwise.

Status openni::VideoStream::readFrame (VideoFrameRef * pFrame)

Read the next frame from this video stream, delivered as a **VideoFrameRef**. This is the primary method for manually obtaining frames of video data. If no new frame is available, the call will block until one is available. To avoid blocking, use **VideoStream::Listener** to implement an event driven architecture. Another alternative is to use **OpenNI::waitForAnyStream()** to wait for new frames from several streams.

Parameters:

[out] **pFrame** Pointer to a **VideoFrameRef** object to hold the reference to the new frame.

Returns:

Status code to indicated success or failure of this function.

void openni::VideoStream::removeNewFrameListener (NewFrameLis

Removes a Listener from this video stream list. The listener removed will no longer receive new frame events from this stream.

Parameters:

[in] **pListener** Pointer to the listener object to be removed.

Status **openni::VideoStream::resetCropping()** [inline]

Disables cropping.

Returns:

Status code indicating success or failure of this operation.

Status **openni::VideoStream::setCropping(int originX, int originY, int width, int height)** [inline]

Changes the cropping settings for this stream. You can use the **isCroppingSupported()** function to make sure cropping is supported before calling this function.

Parameters:

[in] **originX** New X coordinate of the upper left corner of the cropping window.

[in] **originY** New Y coordinate of the upper left corner of the cropping window.

[in] **width** New horizontal width for the cropping window, in pixels.

[in] **height** New vertical height for the cropping window, in pixels.

Returns:

Status code indicating success or failure of this operation.

Status **openni::VideoStream:: setFrameBuffersAllocator(FrameAlloc**

Sets the frame buffers allocator for this video stream.

Parameters:

[in] **pAllocator** Pointer to the frame buffers allocator object.
Pass NULL to return to default frame allocator.

Returns:

ONI_STATUS_OUT_OF_FLOW The frame buffers allocator cannot be set while stream is streaming.

Status **openni::VideoStream::setMirroringEnabled (bool isEnabled)**

Enable or disable mirroring for this stream.

Parameters:

[in] **isEnabled** true to enable mirroring, false to disable it.

Returns:

Status code indicating the success or failure of this operation.

Status **openni::VideoStream::setProperty (int propertyId, const void * data, int dataSize) [inline]**

General function for setting the value of stream specific properties. There are convenience functions available for all commonly used properties, so it is not expected that applications will make direct use of the setProperty function very often.

Parameters:

[in] **propertyId** The numerical ID of the property to be set.
[in] **data** Place to store the data to be written to the

property.

[in] dataSize Size of the data to be written to the property.

Returns:

Status code indicating success or failure of this operation.

template<class T >

**Status openni::VideoStream::setProperty (int propertyId,
const T & value
) [inline]**

Function for setting a value of a stream property using an arbitrary input type. There are convenience functions available for all commonly used properties, so it is not expected that applications will make direct use of this function very often.

Template Parameters:

[in] T Data type of the value to be passed to the property.

Parameters:

[in] propertyId The numerical ID of the property to be set.
[in] value Data to be sent to the property.

Returns:

Status code indicating success or failure of this operation.

**Status openni::VideoStream::setVideoMode (const VideoMode & **

Changes the current video mode of this stream. Recommended practice is to use [Device::getSensorInfo\(\)](#), and then [SensorInfo::getSupportedVideoModes\(\)](#) to obtain a list of valid video mode settings for this stream. Then, pass a valid [VideoMode](#) to [setVideoMode](#) to ensure correct operation.

Parameters:

[in] **videoMode** Desired new video mode for this stream.
returns Status code indicating success or failure of this operation.

Status `openni::VideoStream::start()` [inline]

Starts data generation from this video stream.

`void openni::VideoStream::stop()` [inline]

Stops data generation from this video stream.

Friends And Related Function Documentation

friend class Device [friend]

The documentation for this class was generated from the following file:

- [OpenNI.h](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Members](#)

openni

YUV422DoublePixel

Public Attributes

openni::YUV422DoublePixel Struct Reference

#include <[OpenNI.h](#)>

List of all members.

Public Attributes

uint8_t **u**

uint8_t **v**

uint8_t **y1**

uint8_t **y2**

Detailed Description

Holds the value of two pixels in YUV422 format (Luminance/Chrominance,16-bits/pixel). The first pixel has the values y1, u, v. The second pixel has the values y2, u, v.

Member Data Documentation

`uint8_t openni::YUV422DoublePixel::u`

First chrominance value for two pixels, stored as blue luminance difference signal.

`uint8_t openni::YUV422DoublePixel::v`

Second chrominance value for two pixels, stored as red luminance difference signal.

`uint8_t openni::YUV422DoublePixel::y1`

Overall luminance value of first pixel.

`uint8_t openni::YUV422DoublePixel::y2`

Overall luminance value of second pixel.

The documentation for this struct was generated from the following file:

- [OpenNI.h](#)

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	

Class Index

A | C | D | F | N | O | P | R | S | V | Y

OpenNI::DeviceConnectedListener
([openni](#))

A

OpenNI::DeviceDisconnectedListener
([openni](#))

O

[Array](#) ([openni](#))

DeviceInfo ([openni](#))

O
(op)

C

OpenNI::DeviceStateChangedListener
([openni](#))

P

[CameraSettings](#)
([openni](#))

F

[Playba](#)
(op)

[CoordinateConverter](#)
([openni](#))

VideoStream::FrameAllocator
([openni](#))

R

D

N

Re
(op)

[Device](#) ([openni](#))

VideoStream::NewFrameListener
([openni](#))

A | C | D | F | N | O | P | R | S | V | Y

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files															
Class List	Class Index	Class Members																
All	Functions	Variables	Related Functions															
a	b	c	d	e	f	g	h	i	m	n	o	r	s	u	v	w	y	~

Here is a list of all class members with links to the classes they belong to:

- a -

- addDeviceConnectedListener() : [openni::OpenNI](#)
- addDeviceDisconnectedListener() : [openni::OpenNI](#)
- addDeviceStateChangedListener() : [openni::OpenNI](#)
- addNewFrameListener() : [openni::VideoStream](#)
- allocateFrameBuffer() : [openni::VideoStream::FrameAllocator](#)
- Array() : [openni::Array< T >](#)
- attach() : [openni::Recorder](#)

- b -

- b : [openni::RGB888Pixel](#)
- build : [openni::Version](#)

- c -

- close() : [openni::Device](#)
- convertDepthToColor() : [openni::CoordinateConverter](#)
- convertDepthToWorld() : [openni::CoordinateConverter](#)
- convertWorldToDepth() : [openni::CoordinateConverter](#)
- create() : [openni::Recorder](#) , [openni::VideoStream](#)

- d -

- destroy() : [openni::VideoStream](#) , [openni::Recorder](#)

- Device : **openni::DeviceInfo** , **openni::Device** ,
openni::VideoStream , **openni::Device** ,
openni::PlaybackControl , **openni::SensorInfo**
- DeviceConnectedListener() :
openni::OpenNI::DeviceConnectedListener
- DeviceDisconnectedListener() :
openni::OpenNI::DeviceDisconnectedListener
- DeviceStateChangedListener() :
openni::OpenNI::DeviceStateChangedListener

- e -

- enumerateDevices() : **openni::OpenNI**

- f -

- freeFrameBuffer() : **openni::VideoStream::FrameAllocator**

- g -

- g : **openni::RGB888Pixel**
- getAutoExposureEnabled() : **openni::CameraSettings**
- getAutoWhiteBalanceEnabled() : **openni::CameraSettings**
- getCameraSettings() : **openni::VideoStream**
- getCropOriginX() : **openni::VideoFrameRef**
- getCropOriginY() : **openni::VideoFrameRef**
- getcropping() : **openni::VideoStream**
- getcroppingEnabled() : **openni::VideoFrameRef**
- getData() : **openni::VideoFrameRef**
- getDataSize() : **openni::VideoFrameRef**
- getDepthColorSyncEnabled() : **openni::Device**
- getDeviceInfo() : **openni::Device**
- getExposure() : **openni::CameraSettings**
- getExtendedError() : **openni::OpenNI**
- getFps() : **openni::VideoMode**
- getFrameIndex() : **openni::VideoFrameRef**
- getGain() : **openni::CameraSettings**
- getHeight() : **openni::VideoFrameRef**

- `getHorizontalFieldOfView() : openni::VideoStream`
- `getImageRegistrationMode() : openni::Device`
- `getLogFileName() : openni::OpenNI`
- `getMaxPixelValue() : openni::VideoStream`
- `getMinPixelValue() : openni::VideoStream`
- `getMirroringEnabled() : openni::VideoStream`
- `getName() : openni::DeviceInfo`
- `getNumberOfFrames() : openni::PlaybackControl`
- `getPixelFormat() : openni::VideoMode`
- `getPlaybackControl() : openni::Device`
- `getProperty() : openni::Device , openni::VideoStream`
- `getRepeatEnabled() : openni::PlaybackControl`
- `getResolutionX() : openni::VideoMode`
- `getResolutionY() : openni::VideoMode`
- `getSensorInfo() : openni::VideoStream , openni::Device`
- `getSensorType() : openni::SensorInfo , openni::VideoFrameRef`
- `getSize() : openni::Array< T >`
- `getSpeed() : openni::PlaybackControl`
- `getStrideInBytes() : openni::VideoFrameRef`
- `getSupportedVideoModes() : openni::SensorInfo`
- `getTimestamp() : openni::VideoFrameRef`
- `getUri() : openni::DeviceInfo`
- `getUsbProductId() : openni::DeviceInfo`
- `getUsbVendorId() : openni::DeviceInfo`
- `getVendor() : openni::DeviceInfo`
- `getVersion() : openni::OpenNI`
- `getVerticalFieldOfView() : openni::VideoStream`
- `getVideoMode() : openni::VideoStream ,
openni::VideoFrameRef`
- `getWidth() : openni::VideoFrameRef`

- h -

- `hasSensor() : openni::Device`

- i -

- `initialize() : openni::OpenNI`

- invoke() : **openni::VideoStream** , **openni::Device** , **openni::VideoStream**
- isCommandSupported() : **openni::VideoStream** , **openni::Device**
- isCroppingSupported() : **openni::VideoStream**
- isFile() : **openni::Device**
- isImageRegistrationModeSupported() : **openni::Device**
- isPropertySupported() : **openni::VideoStream** , **openni::Device**
- isValid() : **openni::VideoStream** , **openni::VideoFrameRef** , **openni::CameraSettings** , **openni::Device** , **openni::Recorder** , **openni::PlaybackControl**

- m -

- maintenance : **openni::Version**
- major : **openni::Version**
- minor : **openni::Version**

- n -

- NewFrameListener() : **openni::VideoStream::NewFrameListener**

- o -

- onDeviceConnected() :
openni::OpenNI::DeviceConnectedListener
- onDeviceDisconnected() :
openni::OpenNI::DeviceDisconnectedListener
- onDeviceStateChanged() :
openni::OpenNI::DeviceStateChangedListener
- onNewFrame() : **openni::VideoStream::NewFrameListener**
- open() : **openni::Device**
- OpenNI : **openni::OpenNI::DeviceStateChangedListener** ,
openni::OpenNI::DeviceConnectedListener ,
openni::OpenNI::DeviceDisconnectedListener ,
openni::DeviceInfo
- operator=() : **openni::VideoFrameRef** , **openni::VideoMode**
- operator[]() : **openni::Array< T >**

- r -

- r : **openni::RGB888Pixel**
- readFrame() : **openni::VideoStream**
- Recorder() : **openni::Recorder**
- release() : **openni::VideoFrameRef**
- removeDeviceConnectedListener() : **openni::OpenNI**
- removeDeviceDisconnectedListener() : **openni::OpenNI**
- removeDeviceStateChangedListener() : **openni::OpenNI**
- removeNewFrameListener() : **openni::VideoStream**
- resetCropping() : **openni::VideoStream**

- s -

- seek() : **openni::PlaybackControl**
- SensorInfo : **openni::VideoMode**
- setAutoExposureEnabled() : **openni::CameraSettings**
- setAutoWhiteBalanceEnabled() : **openni::CameraSettings**
- setCropping() : **openni::VideoStream**
- setDepthColorSyncEnabled() : **openni::Device**
- setExposure() : **openni::CameraSettings**
- setFps() : **openni::VideoMode**
- setFrameBuffersAllocator() : **openni::VideoStream**
- setGain() : **openni::CameraSettings**
- setImageRegistrationMode() : **openni::Device**
- setLogAndroidOutput() : **openni::OpenNI**
- setLogConsoleOutput() : **openni::OpenNI**
- setLogFileOutput() : **openni::OpenNI**
- setLogMinSeverity() : **openni::OpenNI**
- setLogOutputFolder() : **openni::OpenNI**
- setMirroringEnabled() : **openni::VideoStream**
- setPixelFormat() : **openni::VideoMode**
- setProperty() : **openni::VideoStream , openni::Device , openni::VideoStream**
- setRepeatEnabled() : **openni::PlaybackControl**
- setResolution() : **openni::VideoMode**
- setSpeed() : **openni::PlaybackControl**
- setVideoMode() : **openni::VideoStream**

- shutdown() : **openni::OpenNI**
- start() : **openni::Recorder** , **openni::VideoStream**
- stop() : **openni::VideoStream** , **openni::Recorder**

- **u** -

- u : **openni::YUV422DoublePixel**

- **v** -

- v : **openni::YUV422DoublePixel**
- VideoFrameRef : **openni::VideoMode** , **openni::VideoFrameRef**
- VideoMode() : **openni::VideoMode**
- VideoStream : **openni::VideoFrameRef** , **openni::SensorInfo** ,
openni::VideoMode , **openni::CameraSettings** ,
openni::VideoStream ,
openni::VideoStream::NewFrameListener ,
openni::VideoStream::FrameAllocator

- **w** -

- waitForAnyStream() : **openni::OpenNI**

- **y** -

- y1 : **openni::YUV422DoublePixel**
- y2 : **openni::YUV422DoublePixel**

- ~ -

- ~Array() : **openni::Array< T >**
- ~Device() : **openni::Device**
- ~DeviceConnectedListener() :
openni::OpenNI::DeviceConnectedListener
- ~DeviceDisconnectedListener() :
openni::OpenNI::DeviceDisconnectedListener
- ~DeviceStateChangedListener() :
openni::OpenNI::DeviceStateChangedListener
- ~FrameAllocator() : **openni::VideoStream::FrameAllocator**

- ~NewFrameListener() : [**openni::VideoStream::NewFrameListener**](#)
 - ~PlaybackControl() : [**openni::PlaybackControl**](#)
 - ~Recorder() : [**openni::Recorder**](#)
 - ~VideoFrameRef() : [**openni::VideoFrameRef**](#)
 - ~VideoStream() : [**openni::VideoStream**](#)
-

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [**doxygen**](#) 1.7.5.1

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		

Namespace List

Here is a list of all namespaces with brief descriptions:

[openni](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1



OpenNI 2.0

Main Page	Namespaces	Classes	Files	
Namespace List	Namespace Members	Classes Typedefs Enumerations Variables		
openni Namespace Reference				

Classes

class	Array
class	CameraSettings
class	CoordinateConverter
class	Device
class	DeviceInfo
class	OpenNI
class	PlaybackControl
class	Recorder
struct	RGB888Pixel
class	SensorInfo
struct	Version
class	VideoFrameRef
class	VideoMode
class	VideoStream
struct	YUV422DoublePixel

TypeDefs

```
typedef uint16_t DepthPixel  
typedef uint16_t Grayscale16Pixel
```

Enumerations

```
enum DeviceState { DEVICE_STATE_OK = 0,
    DEVICE_STATE_ERROR = 1,
    DEVICE_STATE_NOT_READY = 2, DEVICE_STATE_EOF =
3 }
enum ImageRegistrationMode { IMAGE_REGISTRATION_OFF =
0, IMAGE_REGISTRATION_DEPTH_TO_COLOR = 1 }
enum PixelFormat {
    PIXEL_FORMAT_DEPTH_1_MM = 100,
    PIXEL_FORMAT_DEPTH_100 UM = 101,
    PIXEL_FORMAT_SHIFT_9_2 = 102,
    PIXEL_FORMAT_SHIFT_9_3 = 103,
    PIXEL_FORMAT_RGB888 = 200,
    PIXEL_FORMAT_YUV422 = 201, PIXEL_FORMAT_GRAY8 =
202, PIXEL_FORMAT_GRAY16 = 203,
    PIXEL_FORMAT_JPEG = 204, PIXEL_FORMAT_YUYV =
205
}
enum SensorType { SENSOR_IR = 1, SENSOR_COLOR = 2,
SENSOR_DEPTH = 3 }
enum Status {
    STATUS_OK = 0, STATUS_ERROR = 1,
    STATUS_NOT_IMPLEMENTED = 2,
    STATUS_NOT_SUPPORTED = 3,
    STATUS_BAD_PARAMETER = 4,
    STATUS_OUT_OF_FLOW = 5, STATUS_NO_DEVICE = 6,
    STATUS_TIME_OUT = 102
}
```

Variables

```
static const char * ANY_DEVICE = NULL  
static const int TIMEOUT_FOREVER = -1  
static const int TIMEOUT_NONE = 0
```

Detailed Description

openni is the namespace of the entire C++ API of [OpenNI](#)

TypeDef Documentation

typedef uint16_t [openni::DepthPixel](#)

Pixel type used to store depth images.

typedef uint16_t [openni::Grayscale16Pixel](#)

Pixel type used to store IR images.

Enumeration Type Documentation

enum openni::DeviceState

Enumerator:

DEVICE_STATE_OK
DEVICE_STATE_ERROR
DEVICE_STATE_NOT_READY
DEVICE_STATE_EOF

enum openni::ImageRegistrationMode

Enumerator:

IMAGE_REGISTRATION_OFF
IMAGE_REGISTRATION_DEPTH_TO_COLOR

enum openni::PixelFormat

All available formats of the output of a stream

Enumerator:

PIXEL_FORMAT_DEPTH_1_MM
PIXEL_FORMAT_DEPTH_100 UM
PIXEL_FORMAT_SHIFT_9_2
PIXEL_FORMAT_SHIFT_9_3
PIXEL_FORMAT_RGB888
PIXEL_FORMAT_YUV422
PIXEL_FORMAT_GRAY8
PIXEL_FORMAT_GRAY16
PIXEL_FORMAT_JPEG
PIXEL_FORMAT_YUYV

enum openni::SensorType

The source of the stream

Enumerator:

SENSOR_IR
SENSOR_COLOR
SENSOR_DEPTH

enum openni::Status

Possible failure values

Enumerator:

STATUS_OK
STATUS_ERROR
STATUS_NOT_IMPLEMENTED
STATUS_NOT_SUPPORTED
STATUS_BAD_PARAMETER
STATUS_OUT_OF_FLOW
STATUS_NO_DEVICE
STATUS_TIME_OUT

Variable Documentation

const char* openni::ANY_DEVICE = NULL [static]

This special URI can be passed to [Device::open\(\)](#) when the application has no concern for a specific device.

const int openni::TIMEOUT_FOREVER = -1 [static]

const int openni::TIMEOUT_NONE = 0 [static]

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files		
Namespace List		Namespace Members			
All	Variables	Typedefs	Enumerations	Enumerator	
a	d	g	i	p	s t

Here is a list of all namespace members with links to the namespace documentation for each member:

- a -

- ANY_DEVICE : [openni](#)

- d -

- DepthPixel : [openni](#)
- DEVICE_STATE_EOF : [openni](#)
- DEVICE_STATE_ERROR : [openni](#)
- DEVICE_STATE_NOT_READY : [openni](#)
- DEVICE_STATE_OK : [openni](#)
- DeviceState : [openni](#)

- g -

- Grayscale16Pixel : [openni](#)

- i -

- IMAGE_REGISTRATION_DEPTH_TO_COLOR : [openni](#)
- IMAGE_REGISTRATION_OFF : [openni](#)
- ImageRegistrationMode : [openni](#)

- p -

- PIXEL_FORMAT_DEPTH_100 UM : [openni](#)
- PIXEL_FORMAT_DEPTH_1_MM : [openni](#)

- PIXEL_FORMAT_GRAY16 : [openni](#)
- PIXEL_FORMAT_GRAY8 : [openni](#)
- PIXEL_FORMAT_JPEG : [openni](#)
- PIXEL_FORMAT_RGB888 : [openni](#)
- PIXEL_FORMAT_SHIFT_9_2 : [openni](#)
- PIXEL_FORMAT_SHIFT_9_3 : [openni](#)
- PIXEL_FORMAT_YUV422 : [openni](#)
- PIXEL_FORMAT_YUYV : [openni](#)
- PixelFormat : [openni](#)

- s -

- SENSOR_COLOR : [openni](#)
- SENSOR_DEPTH : [openni](#)
- SENSOR_IR : [openni](#)
- SensorType : [openni](#)
- Status : [openni](#)
- STATUS_BAD_PARAMETER : [openni](#)
- STATUS_ERROR : [openni](#)
- STATUS_NO_DEVICE : [openni](#)
- STATUS_NOT_IMPLEMENTED : [openni](#)
- STATUS_NOT_SUPPORTED : [openni](#)
- STATUS_OK : [openni](#)
- STATUS_OUT_OF_FLOW : [openni](#)
- STATUS_TIME_OUT : [openni](#)

- t -

- TIMEOUT_FOREVER : [openni](#)
- TIMEOUT_NONE : [openni](#)

File List

Here is a list of all files with brief descriptions:

[OniEnums.h \[code\]](#)[OpenNI.h \[code\]](#)



OpenNI 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[File List](#)[Namespaces](#) | [Enumerations](#) |
[Variables](#)

OniEnums.h File Reference

[Go to the source code of this file.](#)

Namespaces

namespace **openni**

Enumerations

```
enum  openni::DeviceState { openni::DEVICE_STATE_OK = 0,
                            openni::DEVICE_STATE_ERROR = 1,
                            openni::DEVICE_STATE_NOT_READY = 2,
                            openni::DEVICE_STATE_EOF = 3 }

enum  openni::ImageRegistrationMode {
        openni::IMAGE_REGISTRATION_OFF = 0,
        openni::IMAGE_REGISTRATION_DEPTH_TO_COLOR = 1 }

enum  openni::PixelFormat {
        openni::PIXEL_FORMAT_DEPTH_1_MM = 100,
        openni::PIXEL_FORMAT_DEPTH_100 UM = 101,
        openni::PIXEL_FORMAT_SHIFT_9_2 = 102,
        openni::PIXEL_FORMAT_SHIFT_9_3 = 103,
        openni::PIXEL_FORMAT_RGB888 = 200,
        openni::PIXEL_FORMAT_YUV422 = 201,
        openni::PIXEL_FORMAT_GRAY8 = 202,
        openni::PIXEL_FORMAT_GRAY16 = 203,
        openni::PIXEL_FORMAT_JPEG = 204,
        openni::PIXEL_FORMAT_YUYV = 205
    }

enum  openni::SensorType { openni::SENSOR_IR = 1,
                           openni::SENSOR_COLOR = 2, openni::SENSOR_DEPTH = 3 }

enum  openni::Status {
        openni::STATUS_OK = 0, openni::STATUS_ERROR = 1,
        openni::STATUS_NOT_IMPLEMENTED = 2,
        openni::STATUS_NOT_SUPPORTED = 3,
        openni::STATUS_BAD_PARAMETER = 4,
        openni::STATUS_OUT_OF_FLOW = 5,
        openni::STATUS_NO_DEVICE = 6,
        openni::STATUS_TIME_OUT = 102
    }
```

Variables

```
static const int openni::TIMEOUT_FOREVER = -1
```

```
static const int openni::TIMEOUT_NONE = 0
```

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI.h File Reference

```
#include "OniPlatform.h" #include "OniProperties.h"  
#include "OniEnums.h"  
#include "OniCAPI.h"  
#include "OniCProperties.h"
```

[Go to the source code of this file.](#)

Classes

class	<code>openni::Array< T ></code>
class	<code>openni::CameraSettings</code>
class	<code>openni::CoordinateConverter</code>
class	<code>openni::Device</code>
class	<code>openni::OpenNI::DeviceConnectedListener</code>
class	<code>openni::OpenNI::DeviceDisconnectedListener</code>
class	<code>openni::DeviceInfo</code>
class	<code>openni::OpenNI::DeviceStateChangedListener</code>
class	<code>openni::VideoStream::FrameAllocator</code>
class	<code>openni::VideoStream::NewFrameListener</code>
class	<code>openni::OpenNI</code>
class	<code>openni::PlaybackControl</code>
class	<code>openni::Recorder</code>
struct	<code>openni::RGB888Pixel</code>
class	<code>openni::SensorInfo</code>
struct	<code>openni::Version</code>
class	<code>openni::VideoFrameRef</code>
class	<code>openni::VideoMode</code>
class	<code>openni::VideoStream</code>
struct	<code>openni::YUV422DoublePixel</code>

Namespaces

namespace **openni**

TypeDefs

```
typedef uint16_t openni::DepthPixel  
typedef uint16_t openni::Grayscale16Pixel
```

Variables

```
static const char * openni::ANY_DEVICE = NULL
```

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI.h

Go to the documentation of this file.

```
00001 /*****
* ****
00002 *
*
00003 *   OpenNI 2.x Alpha
*
00004 *   Copyright (C) 2012 PrimeSense Ltd.
*
00005 *
*
00006 *   This file is part of OpenNI.
*
00007 *
*
00008 *   Licensed under the Apache License, Version
n 2.0 (the "License");
00009 *   you may not use this file except in compliance
with the License.
00010 *   You may obtain a copy of the License at
*
00011 *
*
00012 *       http://www.apache.org/licenses/LICENSE-2.0
*
00013 *
*
00014 *   Unless required by applicable law or agree
```

```
ed to in writing, software      *
00015 * distributed under the License is distribu-
ted on an "AS IS" BASIS,          *
00016 * WITHOUT WARRANTIES OR CONDITIONS OF ANY K-
IND, either express or implied.  *
00017 * See the License for the specific language
governing permissions and      *
00018 * limitations under the License.
                                *

00019 *
                                *

00020 ****
***** /
00021 #ifndef _OPENNI_H_
00022 #define _OPENNI_H_
00023
00024 #include "OniPlatform.h"
00025 #include "OniProperties.h"
00026 #include "OniEnums.h"
00027
00028 #include "OniCAPI.h"
00029 #include "OniCProperties.h"
00030
00034 namespace openni
00035 {
00036
00038 typedef uint16_t           DepthPixel;
00039
00041 typedef uint16_t           Grayscale16P-
ixel;
00042
00043 // structs
00045 typedef struct
00046 {
00048     int major;
00050     int minor;
00052     int maintenance;
```

```
00054     int build;
00055 } Version;
00056
00058 typedef struct
00059 {
00060     /* Red value of this pixel. */
00061     uint8_t r;
00062     /* Green value of this pixel. */
00063     uint8_t g;
00064     /* Blue value of this pixel. */
00065     uint8_t b;
00066 } RGB888Pixel;
00067
00073 typedef struct
00074 {
00076     uint8_t u;
00078     uint8_t y1;
00080     uint8_t v;
00082     uint8_t y2;
00083 } YUV422DoublePixel;
00084
00086 #if ONI_PLATFORM != ONI_PLATFORM_WIN32
00087 #pragma GCC diagnostic ignored "-Wunused-variable"
00088 #pragma GCC diagnostic push
00089 #endif
00090 static const char* ANY_DEVICE = NULL;
00091 #if ONI_PLATFORM != ONI_PLATFORM_WIN32
00092 #pragma GCC diagnostic pop
00093 #endif
00094
00099 template<class T>
00100 class Array
00101 {
00102 public:
00106     Array() : m_data(NULL), m_count(0), m_owner(false) {}
```

```
00107
00115     Array(const T* data, int count) : m_owne
r(false) { _setData(data, count); }
00116
00120     ~Array()
00121     {
00122         clear();
00123     }
00124
00129     int getSize() const { return m_count; }
00130
00134     const T& operator[](int index) const {re
turn m_data[index];}
00135
00146     void _setData(const T* data, int count,
bool isOwner = false)
00147     {
00148         clear();
00149         m_count = count;
00150         m_owner = isOwner;
00151         if (!isOwner)
00152         {
00153             m_data = data;
00154         }
00155         else
00156         {
00157             m_data = new T[count];
00158             memcpy((void*)m_data, data, coun
t * sizeof(T));
00159         }
00160     }
00161
00162 private:
00163     Array(const Array<T>&);
00164     Array<T>& operator=(const Array<T>&);
00165
00166     void clear()
```

```
00167     {
00168         if (m_owner && m_data != NULL)
00169             delete []m_data;
00170         m_owner = false;
00171         m_data = NULL;
00172         m_count = 0;
00173     }
00174
00175     const T* m_data;
00176     int m_count;
00177     bool m_owner;
00178 };
00179
00180 // Forward declaration of all
00181 class SensorInfo;
00182 class VideoStream;
00183 class VideoFrameRef;
00184 class Device;
00185 class OpenNI;
00186 class CameraSettings;
00187 class PlaybackControl;
00188
00203 class VideoMode : private OniVideoMode
00204 {
00205     public:
00211     VideoMode()
00212     {}
00213
00219     VideoMode(const VideoMode& other)
00220     {
00221         *this = other;
00222     }
00223
00230     VideoMode& operator=(const VideoMode& other)
00231     {
00232         setPixelFormat(other.getPixelFormat()
```

```
));
00233         setResolution(other.getResolutionX()
, other.getResolutionY());
00234         setFps(other.getFps());
00235
00236         return *this;
00237     }
00238
00243     PixelFormat getPixelFormat() const { ret
urn (PixelFormat)pixelFormat; }
00244
00249     int getResolutionX() const { return reso
lutionX; }
00250
00255     int getResolutionY() const {return resol
utionY; }
00256
00261     int getFps() const { return fps; }
00262
00269     void setPixelFormat(PixelFormat format)
{ this->pixelFormat = (OniPixelFormat)format; }
00270
00278     void setResolution(int resolutionX, int
resolutionY)
00279     {
00280         this->resolutionX = resolutionX;
00281         this->resolutionY = resolutionY;
00282     }
00283
00290     void setFps(int fps) { this->fps = fps;
}
00291
00292     friend class SensorInfo;
00293     friend class VideoStream;
00294     friend class VideoFrameRef;
00295 };
00296
```

```
00314 class SensorInfo
00315 {
00316 public:
00321     SensorType getSensorType() const { return
00322         (SensorType)m_pInfo->sensorType; }
00322
00330     const Array<VideoMode>& getSupportedVide
00330     oModes() const { return m_videoModes; }
00331
00332 private:
00333     SensorInfo(const SensorInfo&);
00334     SensorInfo& operator=(const SensorInfo&)
00334 ;
00335
00336     SensorInfo() : m_pInfo(NULL), m_videoMod
00336     es(NULL, 0) {}
00337
00338     SensorInfo(const OniSensorInfo* pInfo) :
00338     m_pInfo(NULL), m_videoModes(NULL, 0)
00339     {
00340         _setInternal(pInfo);
00341     }
00342
00343     void _setInternal(const OniSensorInfo* p
00343     Info)
00344     {
00345         m_pInfo = pInfo;
00346         if (pInfo == NULL)
00347         {
00348             m_videoModes._setData(NULL, 0);
00349         }
00350         else
00351         {
00352             m_videoModes._setData(static_cas
00352 t<VideoMode*>(pInfo->pSupportedVideoModes), pInfo-
00352 >numSupportedVideoModes);
00353         }
00353 }
```

```
00354     }
00355
00356     const OniSensorInfo* m_pInfo;
00357     Array<VideoMode> m_videoModes;
00358
00359     friend class VideoStream;
00360     friend class Device;
00361 };
00362
00372 class DeviceInfo : private OniDeviceInfo
00373 {
00374 public:
00379     const char* getUri() const { return uri;
00380     }
00381     const char* getVendor() const { return v
endor; }
00383     const char* getName() const { return nam
e; }
00385     uint16_t getUsbVendorId() const { return
usbVendorId; }
00387     uint16_t getUsbProductId() const { return
usbProductId; }
00388
00389     friend class Device;
00390     friend class OpenNI;
00391 };
00392
00406 class VideoFrameRef
00407 {
00408 public:
00413     VideoFrameRef()
00414     {
00415         m_pFrame = NULL;
00416     }
00417
00421     ~VideoFrameRef()
00422     {
```

```
00423         release();
00424     }
00425
00431     VideoFrameRef(const VideoFrameRef& other
00432         ) : m_pFrame(NULL)
00432     {
00433         _setFrame(other.m_pFrame);
00434     }
00435
00441     VideoFrameRef& operator=(const VideoFram
eRef& other)
00442     {
00443         _setFrame(other.m_pFrame);
00444         return *this;
00445     }
00446
00452     inline int getDataSize() const
00453     {
00454         return m_pFrame->dataSize;
00455     }
00456
00462     inline const void* getData() const
00463     {
00464         return m_pFrame->data;
00465     }
00466
00473     inline SensorType getSensorType() const
00474     {
00475         return (SensorType)m_pFrame->sensort
ype;
00476     }
00477
00485     inline const VideoMode& getVideoMode() c
onst
00486     {
00487         return static_cast<const VideoMode&>
(m_pFrame->videoMode);
```

```
00488     }
00489
00497     inline uint64_t getTimestamp() const
00498     {
00499         return m_pFrame->timestamp;
00500     }
00501
00512     inline int getFrameIndex() const
00513     {
00514         return m_pFrame->frameIndex;
00515     }
00516
00523     inline int getWidth() const
00524     {
00525         return m_pFrame->width;
00526     }
00527
00533     inline int getHeight() const
00534     {
00535         return m_pFrame->height;
00536     }
00537
00542     inline bool getCroppingEnabled() const
00543     {
00544         return m_pFrame->croppingEnabled ==
TRUE;
00545     }
00546
00551     inline int getCropOriginX() const
00552     {
00553         return m_pFrame->cropOriginX;
00554     }
00555
00560     inline int getCropOriginY() const
00561     {
00562         return m_pFrame->cropOriginY;
00563     }
```

```
00564
00570     inline int getStrideInBytes() const
00571     {
00572         return m_pFrame->stride;
00573     }
00574
00578     inline bool isValid() const
00579     {
00580         return m_pFrame != NULL;
00581     }
00582
00587     void release()
00588     {
00589         if (m_pFrame != NULL)
00590         {
00591             oniFrameRelease(m_pFrame);
00592             m_pFrame = NULL;
00593         }
00594     }
00595
00597     void _setFrame(OniFrame* pFrame)
00598     {
00599         setReference(pFrame);
00600         if (pFrame != NULL)
00601         {
00602             oniFrameAddRef(pFrame);
00603         }
00604     }
00605
00607     OniFrame* _getFrame()
00608     {
00609         return m_pFrame;
00610     }
00611
00612 private:
00613     friend class VideoStream;
00614     inline void setReference(OniFrame* pFram
```

```
e)
00615     {
00616         // Initial - don't addref. This is t
00617         he reference from OpenNI
00618         release();
00619         m_pFrame = pFrame;
00620     }
00621     OniFrame* m_pFrame; // const!?
00622 };
00623
00645 class VideoStream
00646 {
00647 public:
00655     class NewFrameListener
00656     {
00657         public:
00661             NewFrameListener() : m_callbackHndl
e(NULL)
00662             {
00663             }
00664
00665             virtual ~NewFrameListener()
00666             {
00667             }
00668
00672             virtual void onNewFrame(VideoStream&
) = 0;
00673
00674     private:
00675         friend class VideoStream;
00676
00677         static void ONI_CALLBACK_TYPE callba
ck(OniStreamHandle streamHandle, void* pCookie)
00678         {
00679             NewFrameListener* pListener = (N
ewFrameListener*)pCookie;
```

```
00680             VideoStream stream;
00681             stream._setHandle(streamHandle);
00682             pListener->onNewFrame(stream);
00683             stream._setHandle(NULL);
00684         }
00685         OniCallbackHandle m_callbackHandle;
00686     };
00687
00688     class FrameAllocator
00689     {
00690     public:
00691         virtual ~FrameAllocator() {}
00692         virtual void* allocateFrameBuffer(int
00693             size) = 0;
00693         virtual void freeFrameBuffer(void* d
00694             ata) = 0;
00694
00695     private:
00696         friend class VideoStream;
00697
00698         static void* ONI_CALLBACK_TYPE alloc
00699             ateFrameBufferCallback(int size, void* pCookie)
00699         {
00700             FrameAllocator* pThis = (FrameAl
00701             locator*)pCookie;
00701             return pThis->allocateFrameBuffer
00702             (size);
00702
00703
00704         static void ONI_CALLBACK_TYPE freeFr
00705             ameBufferCallback(void* data, void* pCookie)
00705         {
00706             FrameAllocator* pThis = (FrameAl
00707             locator*)pCookie;
00707             pThis->freeFrameBuffer(data);
00708         }
00709     };

```

```
00710
00715      VideoStream() : m_stream(NULL), m_sensor
Info(), m_pCameraSettings(NULL), m_isOwner(true)
00716      {}
00717
00722      explicit VideoStream(OniStreamHandle han
dle) : m_stream(NULL), m_sensorInfo(), m_pCameraSe
ttings(NULL), m_isOwner(false)
00723      {
00724          _setHandle(handle);
00725      }
00726
00731      ~VideoStream()
00732      {
00733          destroy();
00734      }
00735
00740      bool isValid() const
00741      {
00742          return m_stream != NULL;
00743      }
00744
00754      inline Status create(const Device& devic
e, SensorType sensorType);
00755
00761      inline void destroy();
00762
00771      const SensorInfo& getSensorInfo() const
00772      {
00773          return m_sensorInfo;
00774      }
00775
00779      Status start()
00780      {
00781          if (!isValid())
00782          {
00783              return STATUS_ERROR;
```

```
00784         }
00785
00786         return (Status)onInputStreamStart(m_stre
00787 am);
00787     }
00788
00792     void stop()
00793     {
00794         if (!isValid())
00795         {
00796             return;
00797         }
00798
00799         onInputStreamStop(m_stream);
00800     }
00801
00812     Status readFrame(VideoFrameRef* pFrame)
00813     {
00814         if (!isValid())
00815         {
00816             return STATUS_ERROR;
00817         }
00818
00819         OniFrame* pOniFrame;
00820         Status rc = (Status)onInputStreamReadFra
00821 me(m_stream, &pOniFrame);
00822
00823         pFrame->setReference(pOniFrame);
00824         return rc;
00824     }
00825
00833     Status addNewFrameListener(NewFrameListe
00834 ner* pListener)
00834     {
00835         if (!isValid())
00836         {
00837             return STATUS_ERROR;
```

```
00838         }
00839
00840         return (Status)onInputStreamRegisterNewF
00841             rameCallback(m_stream, pListener->callback, pListe
00842             ner, &pListener->m_callbackHandle);
00843         }
00844
00845         void removeNewFrameListener(NewFrameList
00846             ener* pListener)
00847         {
00848             if (!isValid())
00849             {
00850                 return;
00851             }
00852
00853
00854             onInputStreamUnregisterNewFrameCallback(
00855             m_stream, pListener->m_callbackHandle);
00856             pListener->m_callbackHandle = NULL;
00857         }
00858
00859         Status setFrameBuffersAllocator(FrameAll
00860             oocator* pAllocator)
00861         {
00862             if (!isValid())
00863             {
00864                 return STATUS_ERROR;
00865             }
00866
00867             if (pAllocator == NULL)
00868             {
00869                 return (Status)onInputStreamSetFrame
00870                 BuffersAllocator(m_stream, NULL, NULL, NULL);
00871             }
00872             else
00873             {
00874                 return (Status)onInputStreamSetFrame
00875                 BuffersAllocator(m_stream, pAllocator->allocateFra
```

```
meBufferCallback, pAllocator->freeFrameBufferCallback, pAllocator);
00877         }
00878     }
00879
00884     OniStreamHandle _getHandle() const
00885     {
00886         return m_stream;
00887     }
00888
00893     CameraSettings* getCameraSettings() {return m_pCameraSettings;}
00894
00905     Status getProperty(int propertyId, void* data, int* dataSize) const
00906     {
00907         if (!isValid())
00908         {
00909             return STATUS_ERROR;
00910         }
00911
00912         return (Status)onistreamGetProperty(
00913             m_stream, propertyId, data, dataSize);
00914
00925     Status setProperty(int propertyId, const void* data, int dataSize)
00926     {
00927         if (!isValid())
00928         {
00929             return STATUS_ERROR;
00930         }
00931
00932         return (Status)onistreamSetProperty(
00933             m_stream, propertyId, data, dataSize);
00934     }
```

```
00941     VideoMode getVideoMode() const
00942     {
00943         VideoMode videoMode;
00944         getProperty<OniVideoMode>(STREAM_PROPERTY_VIDEO_MODE, static_cast<OniVideoMode*>(&videoMode));
00945         return videoMode;
00946     }
00947
00956     Status setVideoMode(const VideoMode& videoMode)
00957     {
00958         return setProperty<OniVideoMode>(STREAM_PROPERTY_VIDEO_MODE, static_cast<const OniVideoMode&>(videoMode));
00959     }
00960
00966     int getMaxPixelValue() const
00967     {
00968         int maxValue;
00969         Status rc = getProperty<int>(STREAM_PROPERTY_MAX_VALUE, &maxValue);
00970         if (rc != STATUS_OK)
00971         {
00972             return 0;
00973         }
00974         return maxValue;
00975     }
00976
00982     int getMinPixelValue() const
00983     {
00984         int minValue;
00985         Status rc = getProperty<int>(STREAM_PROPERTY_MIN_VALUE, &minValue);
00986         if (rc != STATUS_OK)
00987         {
00988             return 0;
```

```
00989         }
00990         return minValue;
00991     }
00992
00997     bool isCroppingSupported() const
00998     {
00999         return isPropertySupported(STREAM_PROPERTY_CROPPING);
01000     }
01001
01010     bool getCropping(int* pOriginX, int* pOriginY, int* pWidth, int* pHeight) const
01011     {
01012         OniCropping cropping;
01013         bool enabled = false;
01014
01015         Status rc = getProperty<OniCropping>(STREAM_PROPERTY_CROPPING, &cropping);
01016
01017         if (rc == STATUS_OK)
01018         {
01019             *pOriginX = cropping.originX;
01020             *pOriginY = cropping.originY;
01021             *pWidth = cropping.width;
01022             *pHeight = cropping.height;
01023             enabled = (cropping.enabled == TRUE);
01024         }
01025
01026         return enabled;
01027     }
01028
01038     Status setCropping(int originX, int originY, int width, int height)
01039     {
01040         OniCropping cropping;
01041         cropping.enabled = true;
```

```
01042         cropping.originX = originX;
01043         cropping.originY = originY;
01044         cropping.width = width;
01045         cropping.height = height;
01046         return setProperty<OniCropping>(STREAM_PROPERTY_CROPPING, cropping);
01047     }
01048
01053     Status resetCropping()
01054     {
01055         OniCropping cropping;
01056         cropping.enabled = false;
01057         return setProperty<OniCropping>(STREAM_PROPERTY_CROPPING, cropping);
01058     }
01059
01064     bool getMirroringEnabled() const
01065     {
01066         OniBool enabled;
01067         Status rc = getProperty<OniBool>(STREAM_PROPERTY_MIRRORING, &enabled);
01068         if (rc != STATUS_OK)
01069         {
01070             return false;
01071         }
01072         return enabled == TRUE;
01073     }
01074
01080     Status setMirroringEnabled(bool isEnabled)
01081     {
01082         return setProperty<OniBool>(STREAM_PROPERTY_MIRRORING, isEnabled ? TRUE : FALSE);
01083     }
01084
01089     float getHorizontalFieldOfView() const
01090     {
```

```
01091         float horizontal = 0;
01092         getProperty<float>(STREAM_PROPERTY_HORIZONTAL_FOV, &horizontal);
01093         return horizontal;
01094     }
01095
01100     float getVerticalFieldOfView() const
01101     {
01102         float vertical = 0;
01103         getProperty<float>(STREAM_PROPERTY_VERTICAL_FOV, &vertical);
01104         return vertical;
01105     }
01106
01116     template <class T>
01117     Status setProperty(int propertyId, const
01118     T& value)
01119     {
01120         return setProperty(propertyId, &value,
01121         sizeof(T));
01120     }
01121
01131     template <class T>
01132     Status getProperty(int propertyId, T* va
01133     lue) const
01134     {
01135         int size = sizeof(T);
01136         return getProperty(propertyId, value,
01136         &size);
01136     }
01137
01143     bool isPropertySupported(int propertyId)
01144     const
01144     {
01145         if (!isValid())
01146         {
01147             return false;
```

```
01148         }
01149
01150         return oniStreamIsPropertySupported(
01151             m_stream, propertyId) == TRUE;
01152     }
01153
01154     Status invoke(int commandId, void* data,
01155     int dataSize)
01156     {
01157         if (!isValid())
01158         {
01159             return STATUS_ERROR;
01160         }
01161
01162         return (Status)onIStreamInvoke(m_
01163             stream, commandId, data, dataSize);
01164     }
01165
01166     template <class T>
01167     Status invoke(int commandId, T& value)
01168     {
01169         return invoke(commandId, &value, siz_
01170             eof(T));
01171     }
01172
01173     bool isCommandSupported(int commandId) c
01174         onst
01175     {
01176         if (!isValid())
01177         {
01178             return false;
01179         }
01180
01181         return (Status)onIStreamIsCommandSup
01182             ported(m_stream, commandId) == TRUE;
01183     }
01184
```

```
01202 private:  
01203     friend class Device;  
01204  
01205     void _setHandle(OniStreamHandle stream)  
01206     {  
01207         m_sensorInfo._setInternal(NULL);  
01208         m_stream = stream;  
01209  
01210         if (stream != NULL)  
01211         {  
01212             m_sensorInfo._setInternal(oniStr  
eamGetSensorInfo(m_stream));  
01213         }  
01214     }  
01215  
01216 private:  
01217     VideoStream(const VideoStream& other);  
01218     VideoStream& operator=(const VideoStream  
& other);  
01219  
01220     OniStreamHandle m_stream;  
01221     SensorInfo m_sensorInfo;  
01222     CameraSettings* m_pCameraSettings;  
01223     bool m_isOwner;  
01224 };  
01225  
01242 class Device  
01243 {  
01244 public:  
01249     Device() : m_pPlaybackControl(NULL), m_d  
evice(NULL), m_isOwner(true)  
01250     {  
01251         clearSensors();  
01252     }  
01253  
01258     explicit Device(OniDeviceHandle handle)  
: m_pPlaybackControl(NULL), m_device(NULL), m_isOw
```

```
ner(false)
01259      {
01260          _setHandle(handle);
01261      }
01262
01267      ~Device()
01268      {
01269          if (m_device != NULL)
01270          {
01271              close();
01272          }
01273      }
01274
01304      inline Status open(const char* uri);
01305
01311      inline void close();
01312
01322      const DeviceInfo& getDeviceInfo() const
01323      {
01324          return mDeviceInfo;
01325      }
01326
01334      bool hasSensor(SensorType sensorType)
01335      {
01336          int i;
01337          for (i = 0; (i < ONI_MAX_SENSORS) &&
01338              (m_aSensorInfo[i].m_pInfo != NULL); ++i)
01339          {
01340              if (m_aSensorInfo[i].getSensorTy
01341                  pe() == sensorType)
01342              {
01343                  return true;
01344              }
01345          if (i == ONI_MAX_SENSORS)
01346          {
```

```
01347             return false;
01348         }
01349
01350         const OniSensorInfo* pInfo = oniDeviceGetSensorInfo(m_device, (OniSensorType)sensorType);
01351
01352         if (pInfo == NULL)
01353     {
01354             return false;
01355     }
01356
01357     m_aSensorInfo[i]._setInternal(pInfo)
01358 ;
01359     return true;
01360 }
01361
01362     const SensorInfo* getSensorInfo(SensorType sensorType)
01363 {
01364     int i;
01365     for (i = 0; (i < ONI_MAX_SENSORS) &&
01366 (m_aSensorInfo[i].m_pInfo != NULL); ++i)
01367     {
01368         if (m_aSensorInfo[i].getSensorType() == sensorType)
01369         {
01370             return &m_aSensorInfo[i];
01371         }
01372     }
01373
01374     // not found. check to see we have a
01375     // additional space
01376     if (i == ONI_MAX_SENSORS)
01377     {
01378         return NULL;
01379     }
01380 }
```

```
01384         }
01385
01386         const OniSensorInfo* pInfo = oniDeviceGetSensorInfo(m_device, (OniSensorType)sensorType);
01387         if (pInfo == NULL)
01388         {
01389             return NULL;
01390         }
01391
01392         m_aSensorInfo[i]._setInternal(pInfo)
01393 ;
01394         return &m_aSensorInfo[i];
01395     }
01400     OniDeviceHandle _getHandle() const
01401     {
01402         return m_device;
01403     }
01404
01409     PlaybackControl* getPlaybackControl() {return m_pPlaybackControl;}
01410
01422     Status getProperty(int propertyId, void* data, int* dataSize) const
01423     {
01424         return (Status)oniDeviceGetProperty(
01425             m_device, propertyId, data, dataSize);
01426     }
01438     Status setProperty(int propertyId, const void* data, int dataSize)
01439     {
01440         return (Status)oniDeviceSetProperty(
01441             m_device, propertyId, data, dataSize);
01442     }
01442
```

```
01450     bool isImageRegistrationModeSupported(Im
ageRegistrationMode mode) const
01451     {
01452         return (oniDeviceIsImageRegistration
ModeSupported(m_device, (OniImageRegistrationMode)
mode) == TRUE);
01453     }
01454
01455     ImageRegistrationMode getImageRegistrati
onMode() const
01456     {
01457         ImageRegistrationMode mode;
01458         Status rc = getProperty<ImageRegistrat
ionMode>(DEVICE_PROPERTY_IMAGE_REGISTRATION, &mo
de);
01459         if (rc != STATUS_OK)
01460         {
01461             return IMAGE_REGISTRATION_OFF;
01462         }
01463         return mode;
01464     }
01465
01466     Status setImageRegistrationMode(ImageReg
istrationMode mode)
01467     {
01468         return setProperty<ImageRegistration
Mode>(DEVICE_PROPERTY_IMAGE_REGISTRATION, mode);
01469     }
01470
01471     bool isValid() const
01472     {
01473         return m_device != NULL;
01474     }
01475
01476     bool isFile() const
01477     {
01478         return isPropertySupported(DEVICE_PR
```

```
OPERTY_PLAYBACK_SPEED) &&
01507                      isPropertySupported(DEVICE_PROPE
RTY_PLAYBACK_REPEAT_ENABLED) &&
01508                      isCommandSupported(DEVICE_COMMAN
D_SEEK);
01509      }
01510
01519      Status setDepthColorSyncEnabled(bool isEnabled)
01520      {
01521          Status rc = STATUS_OK;
01522
01523          if (isEnabled)
01524          {
01525              rc = (Status)onDeviceEnableDept
hColorSync(m_device);
01526          }
01527          else
01528          {
01529              onDeviceDisableDepthColorSync(m
_device);
01530          }
01531
01532          return rc;
01533      }
01534
01535      bool getDepthColorSyncEnabled()
01536      {
01537          return onDeviceGetDepthColorSyncEna
bled(m_device) == TRUE;
01538      }
01539
01550      template <class T>
01551      Status setProperty(int propertyId, const
T& value)
01552      {
01553          return setProperty(propertyId, &valu
```

```
0153     e, sizeof(T));
0154 }
0155
0156     template <class T>
0157     Status getProperty(int propertyId, T* value) const
0158     {
0159         int size = sizeof(T);
0160         return getProperty(propertyId, value, &size);
0161     }
0162
0163     bool isPropertySupported(int propertyId)
0164 const
0165     {
0166         return oniDeviceIsPropertySupported(
0167             m_device, propertyId) == TRUE;
0168     }
0169
0170     Status invoke(int commandId, void* data,
0171 int dataSize)
0172     {
0173         return (Status)oniDeviceInvoke(m_device,
0174             commandId, data, dataSize);
0175     }
0176
0177     template <class T>
0178     Status invoke(int propertyId, T& value)
0179     {
0180         return invoke(propertyId, &value, sizeof(T));
0181     }
0182
0183     bool isCommandSupported(int commandId) const
0184     {
0185         return oniDeviceIsCommandSupported(m
```

```
    _device, commandId) == TRUE;  
01619    }  
01620  
01622    inline Status _openEx(const char* uri, const char* mode);  
01623  
01624 private:  
01625    Device(const Device&);  
01626    Device& operator=(const Device&);  
01627  
01628    void clearSensors()  
01629    {  
01630        for (int i = 0; i < ONI_MAX_SENSORS;  
++i)  
01631        {  
01632            m_aSensorInfo[i]._setInternal(NU  
LL);  
01633        }  
01634    }  
01635  
01636    inline Status _setHandle(OniDeviceHandle  
deviceHandle);  
01637  
01638 private:  
01639    PlaybackControl* m_pPlaybackControl;  
01640  
01641    OniDeviceHandle m_device;  
01642    DeviceInfo mDeviceInfo;  
01643    SensorInfo m_aSensorInfo[ONI_MAX_SENSORS  
];  
01644  
01645    bool m_isOwner;  
01646 };  
01647  
01661 class PlaybackControl  
01662 {  
01663 public:
```

```
01664
01670     ~PlaybackControl()
01671     {
01672         detach();
01673     }
01674
01695     float getSpeed() const
01696     {
01697         if (!isValid())
01698         {
01699             return 0.0f;
01700         }
01701         float speed;
01702         Status rc = m_pDevice->getProperty<float>(DEVICE_PROPERTY_PLAYBACK_SPEED, &speed);
01703         if (rc != STATUS_OK)
01704         {
01705             return 1.0f;
01706         }
01707         return speed;
01708     }
01716     Status setSpeed(float speed)
01717     {
01718         if (!isValid())
01719         {
01720             return STATUS_NO_DEVICE;
01721         }
01722         return m_pDevice->setProperty<float>(DEVICE_PROPERTY_PLAYBACK_SPEED, speed);
01723     }
01724
01730     bool getRepeatEnabled() const
01731     {
01732         if (!isValid())
01733         {
01734             return false;
01735         }
```

```
01736
01737         OniBool repeat;
01738         Status rc = m_pDevice->getProperty<0
01739             <niBool>(DEVICE_PROPERTY_PLAYBACK_REPEAT_ENABLED, &
01740             repeat);
01741         if (rc != STATUS_OK)
01742         {
01743             return false;
01744         }
01745     }
01746
01747     Status setRepeatEnabled(bool repeat)
01748     {
01749         if (!isValid())
01750         {
01751             return STATUS_NO_DEVICE;
01752         }
01753         return m_pDevice->setProperty<OniBool>(DEVICE_PROPERTY_PLAYBACK_REPEAT_ENABLED, repeat
01754             ? TRUE : FALSE);
01755     }
01756
01757     Status seek(const VideoStream& stream, i
01758         nt frameIndex)
01759     {
01760         if (!isValid())
01761         {
01762             return STATUS_NO_DEVICE;
01763         }
01764         OniSeek seek;
01765         seek.frameIndex = frameIndex;
01766         seek.stream = stream._getHandle();
01767         return m_pDevice->invoke(DEVICE_COMM
01768             AND_SEEK, seek);
```

```
01785     }
01786
01795     int getNumberOfFrames(const VideoStream&
01796         stream) const
01797     {
01798         int numOfFrames = -1;
01799         Status rc = stream.getProperty<int>(
01800             STREAM_PROPERTY_NUMBER_OF_FRAMES, &numOfFrames);
01801         if (rc != STATUS_OK)
01802         {
01803             return 0;
01804         }
01805
01806     bool isValid() const
01807     {
01808         return m_pDevice != NULL;
01809     }
01810 private:
01811     Status attach(Device* device)
01812     {
01813         if (!device->isValid() || !device->i
01814             sFile())
01815         {
01816             return STATUS_ERROR;
01817         }
01818         detach();
01819         m_pDevice = device;
01820
01821         return STATUS_OK;
01822     }
01823     void detach()
01824     {
01825         m_pDevice = NULL;
01826     }
```

```
01827
01828     friend class Device;
01829     PlaybackControl(Device* pDevice) : m_pDe
01830     vice(NULL)
01831     {
01832         if (pDevice != NULL)
01833         {
01834             attach(pDevice);
01835         }
01836     }
01837     Device* m_pDevice;
01838 };
01839
01840 class CameraSettings
01841 {
01842 public:
01843     // setters
01844     Status setAutoExposureEnabled(bool enabl
01845 ed)
01846     {
01847         return setProperty(STREAM_PROPERTY_A
01848 UTO_EXPOSURE, enabled ? TRUE : FALSE);
01849     }
01850     Status setAutoWhiteBalanceEnabled(bool e
01851 nabled)
01852     {
01853         return setProperty(STREAM_PROPERTY_A
01854 UTO_WHITE_BALANCE, enabled ? TRUE : FALSE);
01855     }
01856
01857     bool getAutoExposureEnabled() const
01858     {
01859         OniBool enabled = FALSE;
01860
01861         Status rc = getProperty(STREAM_PROPE
01862 RTY_AUTO_EXPOSURE, &enabled);
01863 }
```

```
01858         return rc == STATUS_OK && enabled ==
01859             TRUE;
01860     }
01861     bool getAutoWhiteBalanceEnabled() const
01862     {
01863         OniBool enabled = FALSE;
01864
01865         Status rc = getProperty(STREAM_PROPERTY_
01866             AUTO_WHITE_BALANCE, &enabled);
01867
01868         return rc == STATUS_OK && enabled ==
01869             TRUE;
01870     }
01871
01872     Status setGain(int gain)
01873     {
01874         return setProperty(STREAM_PROPERTY_GAIN,
01875             gain);
01876     }
01877     Status setExposure(int exposure)
01878     {
01879         return setProperty(STREAM_PROPERTY_EXPOSURE,
01880             exposure);
01881     }
01882     int getGain()
01883     {
01884         int gain;
01885         Status rc = getProperty(STREAM_PROPERTY_
01886             GAIN, &gain);
01887
01888         if (rc != STATUS_OK)
01889         {
01890             return 100;
01891         }
01892         return gain;
01893     }
01894     int getExposure()
01895     {
01896         int exposure;
```

```
01889         Status rc = getProperty(STREAM_PROPE
RTY_EXPOSURE, &exposure);
01890         if (rc != STATUS_OK)
01891         {
01892             return 0;
01893         }
01894         return exposure;
01895     }
01896
01897     bool isValid() const {return m_pStream !
= NULL;}
01898 private:
01899     template <class T>
01900     Status getProperty(int propertyId, T* va
lue) const
01901     {
01902         if (!isValid()) return STATUS_NOT_SU
PPORTED;
01903
01904         return m_pStream->getProperty<T>(pro
pertyId, value);
01905     }
01906     template <class T>
01907     Status setProperty(int propertyId, const
T& value)
01908     {
01909         if (!isValid()) return STATUS_NOT_SU
PPORTED;
01910
01911         return m_pStream->setProperty<T>(pro
pertyId, value);
01912     }
01913
01914     friend class VideoStream;
01915     CameraSettings(VideoStream* pStream)
01916     {
01917         m_pStream = pStream;
```

```
01918     }
01919
01920     VideoStream* m_pStream;
01921 };
01922
01923
01936 class OpenNI
01937 {
01938 public:
01939
01955     class DeviceConnectedListener
01956     {
01957         public:
01958             DeviceConnectedListener()
01959             {
01960                 m_deviceConnectedCallbacks.deviceConnected = deviceConnectedCallback;
01961                 m_deviceConnectedCallbacks.deviceDisconnected = NULL;
01962                 m_deviceConnectedCallbacks.deviceStateChanged = NULL;
01963                 m_deviceConnectedCallbacksHandle = NULL;
01964             }
01965
01966         virtual ~DeviceConnectedListener()
01967         {
01968         }
01969
01981         virtual void onDeviceConnected(const
01982             DeviceInfo*) = 0;
01982     private:
01983         static void ONI_CALLBACK_TYPE device
01984             ConnectedCallback(const OniDeviceInfo* pInfo, void
01985             * pCookie)
01984         {
01985             DeviceConnectedListener* pListen
```

```
er = (DeviceConnectedListener*)pCookie;
01986         pListener->onDeviceConnected(sta
tic_cast<const DeviceInfo*>(pInfo));
01987     }
01988
01989     friend class OpenNI;
01990     OniDeviceCallbacks m_deviceConnected
Callbacks;
01991     OniCallbackHandle m_deviceConnectedC
allbacksHandle;
01992
01993 };
02010     class DeviceDisconnectedListener
02011 {
02012     public:
02013         DeviceDisconnectedListener()
02014     {
02015         m_deviceDisconnectedCallbacks.de
viceConnected = NULL;
02016         m_deviceDisconnectedCallbacks.de
viceDisconnected = deviceDisconnectedCallback;
02017         m_deviceDisconnectedCallbacks.de
viceStateChanged = NULL;
02018         m_deviceDisconnectedCallbacksHan
dle = NULL;
02019     }
02020
02021     virtual ~DeviceDisconnectedListener(
)
02022     {
02023     }
02024
02033     virtual void onDeviceDisconnected(co
nst DeviceInfo*) = 0;
02034     private:
02035         static void ONI_CALLBACK_TYPE device
DisconnectedCallback(const OniDeviceInfo* pInfo, v
```

```
02035     oid* pCookie)
02036         {
02037             DeviceDisconnectedListener* pListener = (DeviceDisconnectedListener*)pCookie;
02038             pListener->onDeviceDisconnected(
02039                 static_cast<const DeviceInfo*>(pInfo));
02040             }
02041         friend class OpenNI;
02042         OniDeviceCallbacks m_deviceDisconnectedCallbacks;
02043         OniCallbackHandle m_deviceDisconnectCallbacksHandle;
02044     };
02045     class DeviceStateChangedListener
02046     {
02047         public:
02048             DeviceStateChangedListener()
02049             {
02050                 m_deviceStateChangedCallbacks.deviceConnected = NULL;
02051                 m_deviceStateChangedCallbacks.deviceDisconnected = NULL;
02052                 m_deviceStateChangedCallbacks.deviceStateChanged = deviceStateChangedCallback;
02053                 m_deviceStateChangedCallbacksHandle = NULL;
02054             }
02055         virtual ~DeviceStateChangedListener(
02056         )
02057         {
02058         }
02059         virtual void onDeviceStateChanged(const DeviceInfo*, DeviceState) = 0;
02060     private:
```

```
02081         static void ONI_CALLBACK_TYPE device
02082             StateChangedCallback(const OniDeviceInfo* pInfo, OniDeviceState state, void* pCookie)
02083         {
02084             DeviceStateChangedListener* pListener = (DeviceStateChangedListener*)pCookie;
02085             pListener->onDeviceStateChanged(
02086                 static_cast<const DeviceInfo*>(pInfo), DeviceState(state));
02087         }
02088
02089         friend class OpenNI;
02090         OniDeviceCallbacks m_deviceStateChangedCallbacks;
02091         OniCallbackHandle m_deviceStateChangedCallbacksHandle;
02092     };
02093
02094     static Status initialize()
02095     {
02096         return (Status)oniInitialize(ONI_API_VERSION); // provide version of API, to make sure
02097             proper struct sizes are used
02098     }
02099
02100     static void shutdown()
02101     {
02102         oniShutdown();
02103     }
02104
02105     static Version getVersion()
02106     {
02107         OniVersion oniVersion = oniGetVersion();
02108
02109         Version version;
02110         version.major = oniVersion.major;
02111         version.minor = oniVersion.minor;
```

```
02120         version.maintenance = oniVersion.mai
ntenance;
02121         version.build = oniVersion.build;
02122         return version;
02123     }
02124
02125     static const char* getExtendedError()
02126     {
02127         return oniGetExtendedError();
02128     }
02129
02130     static void enumerateDevices(Array<Device
eInfo>* deviceInfoList)
02131     {
02132         OniDeviceInfo* m_pDeviceInfos;
02133         int m_deviceInfoCount;
02134         oniGetDeviceList(&m_pDeviceInfos, &m_
DeviceInfoCount);
02135         deviceInfoList->_setData((DeviceInfo
*)m_pDeviceInfos, m_deviceInfoCount, true);
02136         oniReleaseDeviceList(m_pDeviceInfos)
02137     }
02138
02139
02140     static Status waitForAnyStream(VideoStre
am** pStreams, int streamCount, int* pReadyStreamI
ndex, int timeout = TIMEOUT_FOREVER)
02141     {
02142         static const int ONI_MAX_STREAMS = 5
02143;
02144         OniStreamHandle streams[ONI_MAX_STRE
AMS];
02145
02146         if (streamCount > ONI_MAX_STREAMS)
02147         {
02148             printf("Too many streams for wai
t: %d > %d\n", streamCount, ONI_MAX_STREAMS);
```

```
02166             return STATUS_BAD_PARAMETER;
02167         }
02168
02169         *pReadyStreamIndex = -1;
02170         for (int i = 0; i < streamCount; ++i
02171     )
02172     {
02173         if (pStreams[i] != NULL)
02174         {
02175             streams[i] = pStreams[i]->g
02176             etHandle();
02177         }
02178         else
02179         {
02180             streams[i] = NULL;
02181         }
02182     }
02183     Status rc = (Status)onWaitForAnyStr
02184     eam(streams, streamCount, pReadyStreamIndex, timeo
02185     ut);
02186
02187     return rc;
02188 }
02189
02190 static Status addDeviceConnectedListener(
02191     DeviceConnectedListener* pListener)
02192 {
02193     if (pListener->m_deviceConnectedCall
02194     backsHandle != NULL)
02195     {
02196         return STATUS_ERROR;
02197     }
02198     return (Status)onRegisterDeviceCall
02199     backs(&pListener->m_deviceConnectedCallbacks, pList
02200     tener, &pListener->m_deviceConnectedCallbacksHandle);
02201 }
```

```
02208     static Status addDeviceDisconnectedListe
02209         ner(DeviceDisconnectedListener* pListener)
02210         {
02211             if (pListener->m_deviceDisconnectedC
02212                 allbacksHandle != NULL)
02213             {
02214                 return STATUS_ERROR;
02215             }
02216         static Status addDeviceStateChangedListe
02217         ner(DeviceStateChangedListener* pListener)
02218         {
02219             if (pListener->m_deviceStateChangedC
02220                 allbacksHandle != NULL)
02221             {
02222                 return STATUS_ERROR;
02223             }
02224         static void removeDeviceConnectedListener
02225         (DeviceConnectedListener* pListener)
02226         {
02227             oniUnregisterDeviceCallbacks(pListen
02228             er->m_deviceConnectedCallbacksHandle);
02229             pListener->m_deviceConnectedCallback
02230             sHandle = NULL;
02231         }
02232         static void removeDeviceDisconnectedList
02233         ener(DeviceDisconnectedListener* pListener)
02234         {
```

```
02252         oniUnregisterDeviceCallbacks(pListener->m_deviceDisconnectedCallbacksHandle);
02253         pListener->m_deviceDisconnectedCallbacksHandle = NULL;
02254     }
02262     static void removeDeviceStateChangedListener(DeviceStateChangedListener* pListener)
02263     {
02264         oniUnregisterDeviceCallbacks(pListener->m_deviceStateChangedCallbacksHandle);
02265         pListener->m_deviceStateChangedCallbacksHandle = NULL;
02266     }
02267
02276     static Status setLogOutputFolder(const char *strLogOutputFolder)
02277     {
02278         return (Status)oniSetLogOutputFolder(strLogOutputFolder);
02279     }
02280
02289     static Status getLogFileName(char *strFileName, int nBufferSize)
02290     {
02292         return (Status)oniGetLogFileName(strFileName, nBufferSize);
02293     }
02294
02304     static Status setLogMinSeverity(int nMinSeverity)
02305     {
02306         return (Status) oniSetLogMinSeverity(nMinSeverity);
02307     }
02308
02317     static Status setLogConsoleOutput(bool bConsoleOutput)
```

```
02318     {
02319         return (Status)onisetLogConsoleOutput(bConsoleOutput);
02320     }
02321
02322     static Status setLogFileOutput(bool bFileOutput)
02323     {
02324         return (Status)onisetLogFileOutput(bFileOutput);
02325     }
02326
02327 #if ONI_PLATFORM == ONI_PLATFORM_ANDROID
02328 _ARM
02329
02330     static Status setLogAndroidOutput(bool bAndroidOutput)
02331     {
02332         return (Status)onisetLogAndroidOutput(bAndroidOutput);
02333     }
02334 #endif
02335
02336 private:
02337     OpenNI()
02338     {
02339     }
02340 };
02341
02342 class CoordinateConverter
02343 {
02344 public:
02345     static Status convertWorldToDepth(const VideoStream& depthStream, float worldX, float worldY, float worldZ, int* pDepthX, int* pDepthY, DepthPixel* pDepthZ)
02346     {
```

```
02407         float depthX, depthY, depthZ;
02408         Status rc = (Status)oniCoordinateCon
02409             verterWorldToDepth(depthStream._getHandle(), world
02410                 X, worldY, worldZ, &depthX, &depthY, &depthZ);
02411                 *pDepthX = (int)depthX;
02412                 *pDepthY = (int)depthY;
02413                 *pDepthZ = (DepthPixel)depthZ;
02414             return rc;
02415     }
02416
02417     static Status convertWorldToDepth(const
02418         VideoStream& depthStream, float worldX, float worl
02419             dY, float worldZ, float* pDepthX, float* pDepthY,
02420             float* pDepthZ)
02421     {
02422         return (Status)oniCoordinateConverte
02423             rWorldToDepth(depthStream._getHandle(), worldX, wo
02424                 rldY, worldZ, pDepthX, pDepthY, pDepthZ);
02425     }
02426
02427     static Status convertDepthToWorld(const
02428         VideoStream& depthStream, int depthX, int depthY,
02429             DepthPixel depthZ, float* pWorldX, float* pWorldY,
02430             float* pWorldZ)
02431     {
02432         return (Status)oniCoordinateConverte
02433             rDepthToWorld(depthStream._getHandle(), float(dept
02434                 hX), float(depthY), float(depthZ), pWorldX, pWorld
02435                 Y, pWorldZ);
02436     }
02437
02438
02439     static Status convertDepthToWorld(const
02440         VideoStream& depthStream, float depthX, float dept
02441             hY, float depthZ, float* pWorldX, float* pWorldY,
02442             float* pWorldZ)
02443     {
02444         return (Status)oniCoordinateConverte
```

```
rDepthToWorld(depthStream._getHandle(), depthX, de  
pthY, depthZ, pWorldX, pWorldY, pWorldZ);  
02458     }  
02459  
02471     static Status convertDepthToColor(const  
VideoStream& depthStream, const VideoStream& color  
Stream, int depthX, int depthY, DepthPixel depthZ,  
int* pColorX, int* pColorY)  
02472     {  
02473         return (Status)oniCoordinateConverte  
rDepthToColor(depthStream._getHandle(), colorStrea  
m._getHandle(), depthX, depthY, depthZ, pColorX, p  
ColorY);  
02474     }  
02475 };  
02476  
02491 class Recorder  
02492 {  
02493 public:  
02498     Recorder() : m_recorder(NULL)  
02499     {  
02500     }  
02501  
02505     ~Recorder()  
02506     {  
02507         destroy();  
02508     }  
02509  
02521     Status create(const char* fileName)  
02522     {  
02523         if (!isValid())  
02524         {  
02525             return (Status)oniCreateRecorder  
(fileName, &m_recorder);  
02526         }  
02527         return STATUS_ERROR;  
02528     }
```

```
02529
02530     bool isValid() const
02531     {
02532         return NULL != getHandle();
02533     }
02534
02535     Status attach(VideoStream& stream, bool
02536     allowLossyCompression = false)
02537     {
02538         if (!isValid() || !stream.isValid())
02539         {
02540             return STATUS_ERROR;
02541         }
02542         return (Status)onRecorderAttachStre
02543         am(
02544             m_recorder,
02545             stream._getHandle(),
02546             allowLossyCompression);
02547     }
02548
02549     Status start()
02550     {
02551         if (!isValid())
02552         {
02553             return STATUS_ERROR;
02554         }
02555         return (Status)onRecorderStart(m_re
02556         corder);
02557     }
02558
02559     void stop()
02560     {
02561         if (isValid())
02562         {
02563             onRecorderStop(m_recorder);
02564         }
02565     }
```

```
02588
02592     void destroy()
02593     {
02594         if (isValid())
02595         {
02596             onRecorderDestroy(&m_recorder);
02597         }
02598     }
02599
02600 private:
02601     Recorder(const Recorder&);
02602     Recorder& operator=(const Recorder&);
02603
02607     OniRecorderHandle getHandle() const
02608     {
02609         return m_recorder;
02610     }
02611
02612
02613     OniRecorderHandle m_recorder;
02614 };
02615
02616 // Implementation
02617 Status VideoStream::create(const Device& device, SensorType sensorType)
02618 {
02619     OniStreamHandle streamHandle;
02620     Status rc = (Status)oniDeviceCreateStrea
m(device._getHandle(), (OniSensorType)sensorType,
&streamHandle);
02621     if (rc != STATUS_OK)
02622     {
02623         return rc;
02624     }
02625
02626     m_isOwner = true;
02627     _setHandle(streamHandle);
```

```
02628
02629     if (isPropertySupported(STREAM_PROPERTY_
AUTO_WHITE_BALANCE) && isPropertySupported(STREAM_
PROPERTY_AUTO_EXPOSURE))
02630     {
02631         m_pCameraSettings = new CameraSetting
gs(this);
02632     }
02633
02634     return STATUS_OK;
02635 }
02636
02637 void VideoStream::destroy()
02638 {
02639     if (!isValid())
02640     {
02641         return;
02642     }
02643
02644     if (m_pCameraSettings != NULL)
02645     {
02646         delete m_pCameraSettings;
02647         m_pCameraSettings = NULL;
02648     }
02649
02650     if (m_stream != NULL)
02651     {
02652         if(m_isOwner)
02653             onStreamDestroy(m_stream);
02654         m_stream = NULL;
02655     }
02656 }
02657
02658 Status Device::open(const char* uri)
02659 {
02660     //If we are not the owners, we stick wit
h our own device
```

```
02661     if(!m_isOwner)
02662     {
02663         if(isValid()){
02664             return STATUS_OK;
02665         }else{
02666             return STATUS_OUT_OF_FLOW;
02667         }
02668     }
02669
02670     OniDeviceHandle deviceHandle;
02671     Status rc = (Status)oniDeviceOpen(uri, &
deviceHandle);
02672     if (rc != STATUS_OK)
02673     {
02674         return rc;
02675     }
02676
02677     _setHandle(deviceHandle);
02678
02679     return STATUS_OK;
02680 }
02681
02682 Status Device::_openEx(const char* uri, const
char* mode)
02683 {
02684     //If we are not the owners, we stick wit
h our own device
02685     if(!m_isOwner)
02686     {
02687         if(isValid()){
02688             return STATUS_OK;
02689         }else{
02690             return STATUS_OUT_OF_FLOW;
02691         }
02692     }
02693
02694     OniDeviceHandle deviceHandle;
```

```
02695     Status rc = (Status)oniDeviceOpenEx(uri,
02696         mode, &deviceHandle);
02697     if (rc != STATUS_OK)
02698     {
02699         return rc;
02700     }
02701     _setHandle(deviceHandle);
02702
02703     return STATUS_OK;
02704 }
02705
02706 Status Device::_setHandle(OniDeviceHandle de
02707 viceHandle)
02708 {
02709     if (m_device == NULL)
02710     {
02711         m_device = deviceHandle;
02712         clearSensors();
02713
02714         oniDeviceGetInfo(m_device, &m_device
02715 Info);
02716
02717         if (isFile())
02718             m_pPlaybackControl = new Playbac
02719 kControl(this);
02720
02721         // Read deviceInfo
02722         return STATUS_OK;
02723     }
02724
02725     return STATUS_OUT_OF_FLOW;
02726 }
02727
```

```
02728 void Device::close()
02729 {
02730     if (m_pPlaybackControl != NULL)
02731     {
02732         delete m_pPlaybackControl;
02733         m_pPlaybackControl = NULL;
02734     }
02735
02736     if (m_device != NULL)
02737     {
02738         if(m_isOwner)
02739         {
02740             onDeviceClose(m_device);
02741         }
02742
02743         m_device = NULL;
02744     }
02745 }
02746
02747
02748 }
02749
02750 #endif // _OPEN_NI_HPP_
```

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > Array			

openni::Array< T > Member List

This is the complete list of members for [openni::Array< T >](#), including all inherited members.

Array()	openni::Array< T > [inline]
Array(const T *data, int count)	openni::Array< T > [inline]
getSize() const	openni::Array< T > [inline]
operator[](int index) const	openni::Array< T > [inline]
~Array()	openni::Array< T > [inline]

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > CameraSettings >			

openni::CameraSettings Member List

This is the complete list of members for [openni::CameraSettings](#), including all inherited members.

getAutoExposureEnabled() const	openni::CameraSettings	[inline]
getAutoWhiteBalanceEnabled() const	openni::CameraSettings	[inline]
getExposure()	openni::CameraSettings	[inline]
getGain()	openni::CameraSettings	[inline]
isValid() const	openni::CameraSettings	[inline]
setAutoExposureEnabled(bool enabled)	openni::CameraSettings	[inline]
setAutoWhiteBalanceEnabled(bool enabled)	openni::CameraSettings	[inline]
setExposure(int exposure)	openni::CameraSettings	[inline]
setGain(int gain)	openni::CameraSettings	[inline]
VideoStream class	openni::CameraSettings	[friend]

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > CoordinateConverter			

openni::CoordinateConverter Member List

This is the complete list of members for [openni::CoordinateConverter](#), including all inherited members.

<code>convertDepthToColor(const VideoStream &depthStream, const VideoStream &colorStream, int depthX, int depthY, DepthPixel depthZ, int *pColorX, int *pColorY)</code>	openni::CoordinateConverter	[inline, static]
<code>convertDepthToWorld(const VideoStream &depthStream, int depthX, int depthY, DepthPixel depthZ, float *pWorldX, float *pWorldY, float *pWorldZ)</code>	openni::CoordinateConverter	[inline, static]
<code>convertDepthToWorld(const VideoStream &depthStream, float depthX, float depthY, float depthZ, float *pWorldX, float *pWorldY, float *pWorldZ)</code>	openni::CoordinateConverter	[inline, static]
<code>convertWorldToDepth(const VideoStream &depthStream, float worldX, float worldY, float worldZ, int *pDepthX, int *pDepthY, DepthPixel *pDepthZ)</code>	openni::CoordinateConverter	[inline, static]

```
convertWorldToDepth(const  
VideoStream &depthStream,  
float worldX, float worldY,  
float worldZ, float *pDepthX,  
float *pDepthY, float  
*pDepthZ)
```

openni::CoordinateConverter

[[inline](#),
[static](#)]

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > Device			

openni::Device Member List

This is the complete list of members for [openni::Device](#), including all inherited members.

close()	openni:
Device()	openni:
Device(OniDeviceHandle handle)	openni:
getDepthColorSyncEnabled()	openni:
getDeviceInfo() const	openni:
getImageRegistrationMode() const	openni:
getPlaybackControl()	openni:
getProperty(int propertyId, void *data, int *dataSize) const	openni:
getProperty(int propertyId, T *value) const	openni:
getSensorInfo(SensorType sensorType)	openni:
hasSensor(SensorType sensorType)	openni:
invoke(int commandId, void *data, int dataSize)	openni:
invoke(int propertyId, T &value)	openni:
isCommandSupported(int commandId) const	openni:
isFile() const	openni:
isImageRegistrationModeSupported(ImageRegistrationMode mode) const	openni:
isPropertySupported(int propertyId) const	openni:
isValid() const	openni:
open(const char *uri)	openni:
setDepthColorSyncEnabled(bool isEnabled)	openni:

setImageRegistrationMode (ImageRegistrationMode mode)	openni:
setProperty (int propertyId, const void *data, int dataSize)	openni:
setProperty (int propertyId, const T &value)	openni:
~Device()	openni:

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	OpenNI	DeviceConnectedListener	

openni::OpenNI::DeviceConnectedListener Member List

This is the complete list of members for [openni::OpenNI::DeviceConnectedListener](#), including all inherited members.

DeviceConnectedListener()	openni::OpenNI::DeviceConnectedList
onDeviceConnected(const DeviceInfo *)=0	openni::OpenNI::DeviceConnectedList
OpenNI class	openni::OpenNI::DeviceConnectedList
~DeviceConnectedListener()	openni::OpenNI::DeviceConnectedList

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	OpenNI	DeviceDisconnectedListener	

openni::OpenNI::DeviceDisconnectedListener Member List

This is the complete list of members for

[**openni::OpenNI::DeviceDisconnectedListener**](#), including all inherited members.

DeviceDisconnectedListener()	openni::OpenNI::DeviceDisconnect
onDeviceDisconnected(const DeviceInfo *)=0	openni::OpenNI::DeviceDisconnect
OpenNI class	openni::OpenNI::DeviceDisconnect
~DeviceDisconnectedListener()	openni::OpenNI::DeviceDisconnect

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > DevicelInfo >			

openni::DevicelInfo Member List

This is the complete list of members for [openni::DevicelInfo](#), including all inherited members.

Device class	openni::DevicelInfo [friend]
getName() const	openni::DevicelInfo [inline]
getUri() const	openni::DevicelInfo [inline]
getUsbProductId() const	openni::DevicelInfo [inline]
getUsbVendorId() const	openni::DevicelInfo [inline]
getVendor() const	openni::DevicelInfo [inline]
OpenNI class	openni::DevicelInfo [friend]

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	OpenNI	DeviceStateChangedListener	

openni::OpenNI::DeviceStateChangedListener Member List

This is the complete list of members for

[**openni::OpenNI::DeviceStateChangedListener**](#), including all

inherited members.

DeviceStateChangedListener()	openni::OpenNI::DeviceStateChangedListener()
onDeviceStateChanged(const DeviceInfo *, DeviceState)=0	openni::OpenNI::DeviceStateChangedListener::onDeviceStateChanged(const DeviceInfo *, DeviceState)=0
OpenNI class	openni::OpenNI::DeviceStateChangedListener::OpenNI class
~DeviceStateChangedListener()	openni::OpenNI::DeviceStateChangedListener::~DeviceStateChangedListener()

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	VideoStream	FrameAllocator	

openni::VideoStream::FrameAllocator Member List

This is the complete list of members for [openni::VideoStream::FrameAllocator](#), including all inherited members.

allocateFrameBuffer(int size)=0	openni::VideoStream::FrameAllocator	[pure virtual]
freeFrameBuffer(void *data)=0	openni::VideoStream::FrameAllocator	[pure virtual]
VideoStream class	openni::VideoStream::FrameAllocator	[friend]
~FrameAllocator()	openni::VideoStream::FrameAllocator	[inlined virtual]

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	VideoStream	NewFrameListener	

openni::VideoStream::NewFrameListener Member List

This is the complete list of members for [openni::VideoStream::NewFrameListener](#), including all inherited members.

NewFrameListener()	openni::VideoStream::NewFrameListener
onNewFrame(VideoStream &)=0	openni::VideoStream::NewFrameListener
VideoStream class	openni::VideoStream::NewFrameListener
~NewFrameListener()	openni::VideoStream::NewFrameListener

OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > OpenNI >			

openni::OpenNI Member List

This is the complete list of members for [openni::OpenNI](#), including all inherited members.

addDeviceConnectedListener (DeviceConnectedListener *pListener)	op
addDeviceDisconnectedListener (DeviceDisconnectedListener *pListener)	op
addDeviceStateChangedListener (DeviceStateChangedListener *pListener)	op
enumerateDevices (Array<DeviceInfo> *deviceInfoList)	op
getExtendedError ()	op
getLogFileName (char *strFileName, int nBufferSize)	op
getVersion ()	op
initialize ()	op
removeDeviceConnectedListener (DeviceConnectedListener *pListener)	op
removeDeviceDisconnectedListener (DeviceDisconnectedListener *pListener)	op
removeDeviceStateChangedListener (DeviceStateChangedListener *pListener)	op
setLogAndroidOutput (bool bAndroidOutput)	op
setLogConsoleOutput (bool bConsoleOutput)	op

setLogFileOutput (bool bLogFileOutput)	op
setLogMinSeverity (int nMinSeverity)	op
setLogOutputFolder (const char *strLogOutputFolder)	op
shutdown()	op
waitForAnyStream (VideoStream **pStreams, int streamCount, int *pReadyStreamIndex, int timeout=TIMEOUT_FOREVER)	op

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > PlaybackControl			

openni::PlaybackControl Member List

This is the complete list of members for [openni::PlaybackControl](#), including all inherited members.

Device class	openni::PlaybackControl [friend]
getNumberOfFrames (const VideoStream &stream) const	openni::PlaybackControl [inline]
getRepeatEnabled () const	openni::PlaybackControl [inline]
getSpeed () const	openni::PlaybackControl [inline]
isValid () const	openni::PlaybackControl [inline]
seek (const VideoStream &stream, int frameIndex)	openni::PlaybackControl [inline]
setRepeatEnabled (bool repeat)	openni::PlaybackControl [inline]
setSpeed (float speed)	openni::PlaybackControl [inline]
~PlaybackControl ()	openni::PlaybackControl [inline]

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > Recorder			

openni::Recorder Member List

This is the complete list of members for [openni::Recorder](#), including all inherited members.

attach(VideoStream &stream, bool allowLossyCompression=false)	openni::Recorder	[inline]
create(const char *fileName)	openni::Recorder	[inline]
destroy()	openni::Recorder	[inline]
isValid() const	openni::Recorder	[inline]
Recorder()	openni::Recorder	[inline]
start()	openni::Recorder	[inline]
stop()	openni::Recorder	[inline]
~Recorder()	openni::Recorder	[inline]

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	RGB888Pixel		

openni::RGB888Pixel Member List

This is the complete list of members for [openni::RGB888Pixel](#), including all inherited members.

[b openni::RGB888Pixel](#)

[g openni::RGB888Pixel](#)

[r openni::RGB888Pixel](#)

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > SensorInfo >			

openni::SensorInfo Member List

This is the complete list of members for [openni::SensorInfo](#), including all inherited members.

Device class	openni::SensorInfo [friend]
getSensorType() const	openni::SensorInfo [inline]
getSupportedVideoModes() const	openni::SensorInfo [inline]
VideoStream class	openni::SensorInfo [friend]



OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > Version >			

openni::Version Member List

This is the complete list of members for [openni::Version](#), including all inherited members.

build	openni::Version
maintenance	openni::Version
major	openni::Version
minor	openni::Version

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni > VideoFrameRef			

openni::VideoFrameRef Member List

This is the complete list of members for [openni::VideoFrameRef](#), including all inherited members.

getCropOriginX() const	openni::VideoFrameRef [inline]
getCropOriginY() const	openni::VideoFrameRef [inline]
getCroppingEnabled() const	openni::VideoFrameRef [inline]
getData() const	openni::VideoFrameRef [inline]
getDataSize() const	openni::VideoFrameRef [inline]
getFrameIndex() const	openni::VideoFrameRef [inline]
getHeight() const	openni::VideoFrameRef [inline]
getSensorType() const	openni::VideoFrameRef [inline]
getStrideInBytes() const	openni::VideoFrameRef [inline]
getTimestamp() const	openni::VideoFrameRef [inline]
getVideoMode() const	openni::VideoFrameRef [inline]
getWidth() const	openni::VideoFrameRef [inline]
isValid() const	openni::VideoFrameRef [inline]
operator=(const VideoFrameRef &other)	openni::VideoFrameRef [inline]
release()	openni::VideoFrameRef [inline]
VideoFrameRef()	openni::VideoFrameRef [inline]
VideoFrameRef(const VideoFrameRef &other)	openni::VideoFrameRef [inline]
VideoStream class	openni::VideoFrameRef [friend]
~VideoFrameRef()	openni::VideoFrameRef [inline]

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	VideoMode		

openni::VideoMode Member List

This is the complete list of members for [openni::VideoMode](#), including all inherited members.

getFps() const	openni::VideoMode [inline]
getPixelFormat() const	openni::VideoMode [inline]
getResolutionX() const	openni::VideoMode [inline]
getResolutionY() const	openni::VideoMode [inline]
operator=(const VideoMode &other)	openni::VideoMode [inline]
SensorInfo class	openni::VideoMode [friend]
setFps(int fps)	openni::VideoMode [inline]
setPixelFormat(PixelFormat format)	openni::VideoMode [inline]
setResolution(int resolutionX, int resolutionY)	openni::VideoMode [inline]
VideoFrameRef class	openni::VideoMode [friend]
VideoMode()	openni::VideoMode [inline]
VideoMode(const VideoMode &other)	openni::VideoMode [inline]
VideoStream class	openni::VideoMode [friend]

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	VideoStream		

openni::VideoStream Member List

This is the complete list of members for [openni::VideoStream](#), including all inherited members.

addNewFrameListener(NewFrameListener *pListener)	openni::VideoStream
create(const Device &device, SensorType sensorType)	openni::VideoStream
destroy()	openni::VideoStream
Device class	openni::VideoStream
getCameraSettings()	openni::VideoStream
getCropping(int *pOriginX, int *pOriginY, int *pWidth, int *pHeight) const	openni::VideoStream
getHorizontalFieldOfView() const	openni::VideoStream
getMaxPixelValue() const	openni::VideoStream
getMinPixelValue() const	openni::VideoStream
getMirroringEnabled() const	openni::VideoStream
getProperty(int propertyId, void *data, int *dataSize) const	openni::VideoStream
getProperty(int propertyId, T *value) const	openni::VideoStream
getSensorInfo() const	openni::VideoStream
getVerticalFieldOfView() const	openni::VideoStream
getVideoMode() const	openni::VideoStream
invoke(int commandId, void *data, int dataSize)	openni::VideoStream
invoke(int commandId, T &value)	openni::VideoStream
isCommandSupported(int commandId) const	openni::VideoStream

<code>isCroppingSupported() const</code>	<code>openni::VideoStream</code>
<code>isPropertySupported(int propertyId) const</code>	<code>openni::VideoStream</code>
<code>isValid() const</code>	<code>openni::VideoStream</code>
<code>readFrame(VideoFrameRef *pFrame)</code>	<code>openni::VideoStream</code>
<code>removeNewFrameListener(NewFrameListener *pListener)</code>	<code>openni::VideoStream</code>
<code>resetCropping()</code>	<code>openni::VideoStream</code>
<code>setCropping(int originX, int originY, int width, int height)</code>	<code>openni::VideoStream</code>
<code>setFrameBuffersAllocator(FrameAllocator *pAllocator)</code>	<code>openni::VideoStream</code>
<code>setMirroringEnabled(bool isEnabled)</code>	<code>openni::VideoStream</code>
<code>setProperty(int propertyId, const void *data, int dataSize)</code>	<code>openni::VideoStream</code>
<code>setProperty(int propertyId, const T &value)</code>	<code>openni::VideoStream</code>
<code>setVideoMode(const VideoMode &videoMode)</code>	<code>openni::VideoStream</code>
<code>start()</code>	<code>openni::VideoStream</code>
<code>stop()</code>	<code>openni::VideoStream</code>
<code>VideoStream()</code>	<code>openni::VideoStream</code>
<code>VideoStream(OniStreamHandle handle)</code>	<code>openni::VideoStream</code>
<code>~VideoStream()</code>	<code>openni::VideoStream</code>

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
openni	YUV422DoublePixel		

openni::YUV422DoublePixel Member List

This is the complete list of members for [openni::YUV422DoublePixel](#), including all inherited members.

u	openni::YUV422DoublePixel
v	openni::YUV422DoublePixel
y1	openni::YUV422DoublePixel
y2	openni::YUV422DoublePixel

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files											
Class List	Class Index	Class Members												
All	Functions	Variables	Related Functions											
a	c	d	e	f	g	h	i	n	o	r	s	v	w	~

- a -

- addDeviceConnectedListener() : [openni::OpenNI](#)
- addDeviceDisconnectedListener() : [openni::OpenNI](#)
- addDeviceStateChangedListener() : [openni::OpenNI](#)
- addNewFrameListener() : [openni::VideoStream](#)
- allocateFrameBuffer() : [openni::VideoStream::FrameAllocator](#)
- Array() : [openni::Array< T >](#)
- attach() : [openni::Recorder](#)

- c -

- close() : [openni::Device](#)
- convertDepthToColor() : [openni::CoordinateConverter](#)
- convertDepthToWorld() : [openni::CoordinateConverter](#)
- convertWorldToDepth() : [openni::CoordinateConverter](#)
- create() : [openni::Recorder](#) , [openni::VideoStream](#)

- d -

- destroy() : [openni::VideoStream](#) , [openni::Recorder](#)
- Device() : [openni::Device](#)
- DeviceConnectedListener() :
[openni::OpenNI::DeviceConnectedListener](#)
- DeviceDisconnectedListener() :
[openni::OpenNI::DeviceDisconnectedListener](#)
- DeviceStateChangedListener() :

openni::OpenNI::DeviceStateChangedListener

- e -

- enumerateDevices() : **openni::OpenNI**

- f -

- freeFrameBuffer() : **openni::VideoStream::FrameAllocator**

- g -

- getAutoExposureEnabled() : **openni::CameraSettings**
- getAutoWhiteBalanceEnabled() : **openni::CameraSettings**
- getCameraSettings() : **openni::VideoStream**
- getCropOriginX() : **openni::VideoFrameRef**
- getCropOriginY() : **openni::VideoFrameRef**
- getCropping() : **openni::VideoStream**
- getCroppingEnabled() : **openni::VideoFrameRef**
- getData() : **openni::VideoFrameRef**
- getDataSize() : **openni::VideoFrameRef**
- getDepthColorSyncEnabled() : **openni::Device**
- getDeviceInfo() : **openni::Device**
- getExposure() : **openni::CameraSettings**
- getExtendedError() : **openni::OpenNI**
- getFps() : **openni::VideoMode**
- getFrameIndex() : **openni::VideoFrameRef**
- getGain() : **openni::CameraSettings**
- getHeight() : **openni::VideoFrameRef**
- getHorizontalFieldOfView() : **openni::VideoStream**
- getImageRegistrationMode() : **openni::Device**
- getLogFileName() : **openni::OpenNI**
- getMaxPixelValue() : **openni::VideoStream**
- getMinPixelValue() : **openni::VideoStream**
- getMirroringEnabled() : **openni::VideoStream**
- getName() : **openni::DeviceInfo**
- getNumberOfFrames() : **openni::PlaybackControl**
- getPixelFormat() : **openni::VideoMode**

- `getPlaybackControl() : openni::Device`
- `getProperty() : openni::Device , openni::VideoStream , openni::Device`
- `getRepeatEnabled() : openni::PlaybackControl`
- `getResolutionX() : openni::VideoMode`
- `getResolutionY() : openni::VideoMode`
- `getSensorInfo() : openni::Device , openni::VideoStream`
- `getSensorType() : openni::SensorInfo , openni::VideoFrameRef`
- `getSize() : openni::Array< T >`
- `getSpeed() : openni::PlaybackControl`
- `getStrideInBytes() : openni::VideoFrameRef`
- `getSupportedVideoModes() : openni::SensorInfo`
- `getTimestamp() : openni::VideoFrameRef`
- `getUri() : openni::DeviceInfo`
- `getUsbProductId() : openni::DeviceInfo`
- `getUsbVendorId() : openni::DeviceInfo`
- `getVendor() : openni::DeviceInfo`
- `getVersion() : openni::OpenNI`
- `getVerticalFieldOfView() : openni::VideoStream`
- `getVideoMode() : openni::VideoFrameRef , openni::VideoStream`
- `getWidth() : openni::VideoFrameRef`

- h -

- `hasSensor() : openni::Device`

- i -

- `initialize() : openni::OpenNI`
- `invoke() : openni::VideoStream , openni::Device , openni::VideoStream`
- `isCommandSupported() : openni::VideoStream , openni::Device`
- `isCroppingSupported() : openni::VideoStream`
- `isFile() : openni::Device`
- `isImageRegistrationModeSupported() : openni::Device`
- `isPropertySupported() : openni::VideoStream , openni::Device`
- `isValid() : openni::VideoStream , openni::VideoFrameRef ,`

**openni::CameraSettings , openni::Device , openni::Recorder ,
openni::PlaybackControl**

- n -

- NewFrameListener() : **openni::VideoStream::NewFrameListener**

- o -

- onDeviceConnected() : **openni::OpenNI::DeviceConnectedListener**
- onDeviceDisconnected() : **openni::OpenNI::DeviceDisconnectedListener**
- onDeviceStateChanged() : **openni::OpenNI::DeviceStateChangedListener**
- onNewFrame() : **openni::VideoStream::NewFrameListener**
- open() : **openni::Device**
- operator=() : **openni::VideoMode , openni::VideoFrameRef**
- operator[]() : **openni::Array< T >**

- r -

- readFrame() : **openni::VideoStream**
- Recorder() : **openni::Recorder**
- release() : **openni::VideoFrameRef**
- removeDeviceConnectedListener() : **openni::OpenNI**
- removeDeviceDisconnectedListener() : **openni::OpenNI**
- removeDeviceStateChangedListener() : **openni::OpenNI**
- removeNewFrameListener() : **openni::VideoStream**
- resetCropping() : **openni::VideoStream**

- s -

- seek() : **openni::PlaybackControl**
- setAutoExposureEnabled() : **openni::CameraSettings**
- setAutoWhiteBalanceEnabled() : **openni::CameraSettings**
- setCropping() : **openni::VideoStream**
- setDepthColorSyncEnabled() : **openni::Device**

- setExposure() : **openni::CameraSettings**
- setFps() : **openni::VideoMode**
- setFrameBuffersAllocator() : **openni::VideoStream**
- setGain() : **openni::CameraSettings**
- setImageRegistrationMode() : **openni::Device**
- setLogAndroidOutput() : **openni::OpenNI**
- setLogConsoleOutput() : **openni::OpenNI**
- setLogFileOutput() : **openni::OpenNI**
- setLogMinSeverity() : **openni::OpenNI**
- setLogOutputFolder() : **openni::OpenNI**
- setMirroringEnabled() : **openni::VideoStream**
- setPixelFormat() : **openni::VideoMode**
- setProperty() : **openni::VideoStream , openni::Device , openni::VideoStream**
- setRepeatEnabled() : **openni::PlaybackControl**
- setResolution() : **openni::VideoMode**
- setSpeed() : **openni::PlaybackControl**
- setVideoMode() : **openni::VideoStream**
- shutdown() : **openni::OpenNI**
- start() : **openni::VideoStream , openni::Recorder**
- stop() : **openni::VideoStream , openni::Recorder**

- V -

- VideoFrameRef() : **openni::VideoFrameRef**
- VideoMode() : **openni::VideoMode**
- VideoStream() : **openni::VideoStream**

- W -

- waitForAnyStream() : **openni::OpenNI**

- ~ -

- ~Array() : **openni::Array< T >**
- ~Device() : **openni::Device**
- ~DeviceConnectedListener() : **openni::OpenNI::DeviceConnectedListener**

- ~DeviceDisconnectedListener() : **openni::OpenNI::DeviceDisconnectedListener**
 - ~DeviceStateChangedListener() : **openni::OpenNI::DeviceStateChangedListener**
 - ~FrameAllocator() : **openni::VideoStream::FrameAllocator**
 - ~NewFrameListener() : **openni::VideoStream::NewFrameListener**
 - ~PlaybackControl() : **openni::PlaybackControl**
 - ~Recorder() : **openni::Recorder**
 - ~VideoFrameRef() : **openni::VideoFrameRef**
 - ~VideoStream() : **openni::VideoStream**
-

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by  1.7.5.1

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
All	Functions	Variables	Related Functions

- b : [openni::RGB888Pixel](#)
- build : [openni::Version](#)
- g : [openni::RGB888Pixel](#)
- maintenance : [openni::Version](#)
- major : [openni::Version](#)
- minor : [openni::Version](#)
- r : [openni::RGB888Pixel](#)
- u : [openni::YUV422DoublePixel](#)
- v : [openni::YUV422DoublePixel](#)
- y1 : [openni::YUV422DoublePixel](#)
- y2 : [openni::YUV422DoublePixel](#)

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	
All	Functions	Variables	Related Functions

- Device : [openni::SensorInfo](#) , [openni::PlaybackControl](#) , [openni::VideoStream](#) , [openni::DeviceInfo](#)
- OpenNI : [openni::DeviceInfo](#) , [openni::OpenNI::DeviceStateChangedListener](#) , [openni::OpenNI::DeviceDisconnectedListener](#) , [openni::OpenNI::DeviceConnectedListener](#)
- SensorInfo : [openni::VideoMode](#)
- VideoFrameRef : [openni::VideoMode](#)
- VideoStream : [openni::CameraSettings](#) , [openni::VideoStream::FrameAllocator](#) , [openni::VideoStream::NewFrameListener](#) , [openni::VideoFrameRef](#) , [openni::SensorInfo](#) , [openni::VideoMode](#)

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files	
Namespace List	Namespace Members			
All	Variables	Typedefs	Enumerations	Enumerator

- ANY_DEVICE : [openni](#)
- TIMEOUT_FOREVER : [openni](#)
- TIMEOUT_NONE : [openni](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files	
Namespace List	Namespace Members			
All	Variables	Typedefs	Enumerations	Enumerator

- DepthPixel : [openni](#)
- Grayscale16Pixel : [openni](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [!\[\]\(90b4db7b80e28f043fe79ccc033163d1_img.jpg\) doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		
All	Variables	Typedefs	Enumerations
			Enumerator

- DeviceState : [openni](#)
- ImageRegistrationMode : [openni](#)
- PixelFormat : [openni](#)
- SensorType : [openni](#)
- Status : [openni](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OpenNI OpenNI 2.0

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		
All	Variables	Typedefs	Enumerations
			Enumerator

- DEVICE_STATE_EOF : [openni](#)
- DEVICE_STATE_ERROR : [openni](#)
- DEVICE_STATE_NOT_READY : [openni](#)
- DEVICE_STATE_OK : [openni](#)
- IMAGE_REGISTRATION_DEPTH_TO_COLOR : [openni](#)
- IMAGE_REGISTRATION_OFF : [openni](#)
- PIXEL_FORMAT_DEPTH_100 UM : [openni](#)
- PIXEL_FORMAT_DEPTH_1_MM : [openni](#)
- PIXEL_FORMAT_GRAY16 : [openni](#)
- PIXEL_FORMAT_GRAY8 : [openni](#)
- PIXEL_FORMAT_JPEG : [openni](#)
- PIXEL_FORMAT_RGB888 : [openni](#)
- PIXEL_FORMAT_SHIFT_9_2 : [openni](#)
- PIXEL_FORMAT_SHIFT_9_3 : [openni](#)
- PIXEL_FORMAT_YUV422 : [openni](#)
- PIXEL_FORMAT_YUYV : [openni](#)
- SENSOR_COLOR : [openni](#)
- SENSOR_DEPTH : [openni](#)
- SENSOR_IR : [openni](#)
- STATUS_BAD_PARAMETER : [openni](#)
- STATUS_ERROR : [openni](#)
- STATUS_NO_DEVICE : [openni](#)
- STATUS_NOT_IMPLEMENTED : [openni](#)
- STATUS_NOT_SUPPORTED : [openni](#)
- STATUS_OK : [openni](#)
- STATUS_OUT_OF_FLOW : [openni](#)
- STATUS_TIME_OUT : [openni](#)

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1

OniEnums.h

Go to the documentation of this file.

```
00001 /*****
* ****
00002 *
*
00003 *   OpenNI 2.x Alpha
*
00004 *   Copyright (C) 2012 PrimeSense Ltd.
*
00005 *
*
00006 *   This file is part of OpenNI.
*
00007 *
*
00008 *   Licensed under the Apache License, Version
n 2.0 (the "License");
00009 *   you may not use this file except in compliance
with the License.
00010 *   You may obtain a copy of the License at
*
00011 *
*
00012 *       http://www.apache.org/licenses/LICENSE-2.0
*
00013 *
*
00014 *   Unless required by applicable law or agree
```

```
ed to in writing, software      *
00015 * distributed under the License is distribu-
ted on an "AS IS" BASIS,          *
00016 * WITHOUT WARRANTIES OR CONDITIONS OF ANY K-
IND, either express or implied.   *
00017 * See the License for the specific language
governing permissions and        *
00018 * limitations under the License.
                                *
00019 *
                                *
00020 ****
***** */
00021 #ifndef _ONI_ENUMS_H_
00022 #define _ONI_ENUMS_H_
00023
00024 namespace openni
00025 {
00026
00028 typedef enum
00029 {
00030     STATUS_OK = 0,
00031     STATUS_ERROR = 1,
00032     STATUS_NOT_IMPLEMENTED = 2,
00033     STATUS_NOT_SUPPORTED = 3,
00034     STATUS_BAD_PARAMETER = 4,
00035     STATUS_OUT_OF_FLOW = 5,
00036     STATUS_NO_DEVICE = 6,
00037     STATUS_TIME_OUT = 102,
00038 } Status;
00039
00041 typedef enum
00042 {
00043     SENSOR_IR = 1,
00044     SENSOR_COLOR = 2,
00045     SENSOR_DEPTH = 3,
00046
```

```
00047 } SensorType;
00048
00050 typedef enum
00051 {
00052     // Depth
00053     PIXEL_FORMAT_DEPTH_1_MM = 100,
00054     PIXEL_FORMAT_DEPTH_100_UM = 101,
00055     PIXEL_FORMAT_SHIFT_9_2 = 102,
00056     PIXEL_FORMAT_SHIFT_9_3 = 103,
00057
00058     // Color
00059     PIXEL_FORMAT_RGB888 = 200,
00060     PIXEL_FORMAT_YUV422 = 201,
00061     PIXEL_FORMAT_GRAY8 = 202,
00062     PIXEL_FORMAT_GRAY16 = 203,
00063     PIXEL_FORMAT_JPEG = 204,
00064     PIXEL_FORMAT_YUYV = 205,
00065 } PixelFormat;
00066
00067 typedef enum
00068 {
00069     DEVICE_STATE_OK      = 0,
00070     DEVICE_STATE_ERROR   = 1,
00071     DEVICE_STATE_NOT_READY = 2,
00072     DEVICE_STATE_EOF     = 3
00073 } DeviceState;
00074
00075 typedef enum
00076 {
00077     IMAGE_REGISTRATION_OFF          = 0,
00078     IMAGE_REGISTRATION_DEPTH_TO_COLOR = 1,
00079 } ImageRegistrationMode;
00080
00081 static const int TIMEOUT_NONE = 0;
00082 static const int TIMEOUT_FOREVER = -1;
00083
00084 } // namespace openni
```

```
00085
00086 #endif // _ONI_ENUMS_H_
```

Generated on Tue Nov 12 2013 16:10:45 for OpenNI 2.0 by [doxygen](#) 1.7.5.1