**Pjsip setup**

- setting openssl : C:\OpenSSL-Win32\bin

- setting Gnu : C:\Program Files (x86)\GnuWin32\bin

- setting cygwin : C:\cygwin

-NDK: CDT - http://download.eclipse.org/tools/cdt/releases/indigo

-SDK: SDK - https://dl-ssl.google.com/android/eclipse/

- SVN - http://subclipse.tigris.org/update\_1.8.x

# - Visual Studio

- setting path

# Installing OpenSSL Library

If TLS support is wanted, then OpenSSL SDK must be installed in the development host.

To install OpenSSL SDK from the Win32 binary distribution:

1. Install OpenSSL SDK to any folder (e.g. C:\OpenSSL)
2. Add OpenSSL DLL location to the system PATH.
3. Add OpenSSL include path to Visual Studio includes search directory. Make sure that OpenSSL header files can be accessed from the program with #include <openssl/ssl.h> construct.
4. Add OpenSSL library path to Visual Studio library search directory. Make sure the following libraries are accessible:
   * libeay32 and ssleay32 (it is recommended to use the same run-time option for PJSIP and the libraries). So, if you compile PJSIP with Multithreaded Debug (/MTd), you may need to use the same run-time option when compiling the library. Please consult the library's doc for more details.

Then to enable TLS transport support in PJSIP, please check [configuring PJSIP with TLS](http://trac.pjsip.org/repos/wiki/TLS).

# Getting Started: Visual Studio Build Configurations

The defaults are:

* **Debug**: multithreaded, statically linked with LIBC, debug (i.e. the **/MTd** flag).
* **Release**: multithreaded, dynamically linked with MSVCRT, release (i.e. the **/MD** flag).

For more precise control use:

* **Debug-Static**: multithreaded, statically linked with LIBC, debug (i.e. the **/MTd** flag).
* **Debug-Dynamic**: multithreaded, dynamically linked with MSVCRT, debug (i.e. the **/MDd** flag).
* **Release-Static**: multithreaded, statically linked with LIBC, release (i.e. the **/MT** flag).
* **Release-Dynamic**: multithreaded, dynamically linked with MSVCRT, release (i.e. the **/MD** flag).

This target naming convention will also apply to Windows Mobile targets.

|  |
| --- |
|  |

# Developing PJSIP with Eclipse

**Table of Contents**

1. [Getting and Installing Eclipse](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#GettingandInstallingEclipse)
2. [Getting and Configuring PJSIP](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#GettingandConfiguringPJSIP)
3. [Setting Up Eclipse](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#SettingUpEclipse)
4. [Importing PJSIP](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#ImportingPJSIP)
5. [Customizing PJSIP](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#CustomizingPJSIP)
6. [Debugging the Project](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#DebuggingtheProject)
7. [Problems and Solutions](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#ProblemsandSolutions)
   1. [I have not been able to set breakpoints](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#Ihavenotbeenabletosetbreakpoints)
   2. [When I step into/over the code, the execution moves erraticly](http://trac.pjsip.org/repos/wiki/Using_Eclipse_with_PJSIP#WhenIstepintooverthecodetheexecutionmoveserraticly)

This short article shows how to use Eclipse for PJSIP development on Linux platform. Using Eclipse one can enjoy Visual Studio like experience in developing PJSIP.

## Getting and Installing Eclipse

1. Download Eclipse for C++ Developers for your platform from [​http://www.eclipse.org/downloads/](http://www.eclipse.org/downloads/).
   * Even on Linux/Ubuntu you'd better off getting and installing Eclipse this way instead of with Package Manager, since as far as I know Eclipse C++ plug-in is not available from the repository (as of Ubuntu 09.10) and installing plug-in manually is rather cumbersome as you'd need to run Eclipse as root.
2. Installation is straightforward, just unpack the tarball containing eclipse directory to your preferred directory, such as**/opt**, and run the **eclipse** executable directly.
   * sudo as root as necessary to write to /opt directory.
3. You will have to install a Java Run-time Environment if you haven't had one. On Ubuntu, just install **default-jre** package.

## Getting and Configuring PJSIP

1. Download and install PJSIP [as usual](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source).
2. Configure PJSIP with minimal setting, e.g.:
3. $ ./aconfigure CFLAGS='-Wno-unused-label'

The reason for the minimal setting is because we want to configure as much possible from Eclipse instead of having to re-run ./aconfigure every time some compilation flags need to be changed. The way to do this (later) is to put your customization in **user.mak** file instead.

1. Do a **make dep**

## Setting Up Eclipse

1. Run Eclipse
2. Configure workspace:
   * Create/select a directory for your workspace the first time you run Eclipse. This directory should be placed outside your PJPROJECT directory.
3. Disable auto-build everytime something changes:
   * Uncheck **Project** --> **Build Automatically**
4. Set proper indentation for PJSIP:
   * Select menu:
     + Linux: **Window** --> **Preferences**
     + Mac: **Eclipse** --> **Preferences**
   * In the **Preferences** dialog, from the tree, select **C/C++** --> **Code Style**
   * in the code style, press **New**:
     + Profile name: **pjsip**
     + Initialize settings with the following profile: **K&R [built-in]**
     + Click OK
   * in the **pjsip** profile editing dialog:
     + Tab policy: **mixed**
     + Uncheck **Use tabs only**
     + Indentation: **4**
     + Tab size: **8**
     + Click OK to close **pjsip** profile editing dialog
   * Click OK to close **Preferences** dialog.
5. To activate Visual Studio style key shortcuts:
   * Select menu:
     + Linux: **Window** --> **Preferences**
     + Mac: **Eclipse** --> **Preferences**
   * In the **Preferences** dialog, from the tree, select **General** --> **Keys**
   * Select **Microsoft Visual Studio** scheme
   * Customize other key shortcuts as wanted (for example, F7 for Build)
6. Disable menubar F10 acceleration key, as we need the F10 to be captured by Eclipse (Visual Studio users use F10 for stepping to next statement during debugging):
   * Ubuntu/Gnome:
     + On Gnome, F10 key will open application's menubar, even if it has been assigned to other thing with Eclipse (for example, Step Over with Visual Studio keys)
     + To disable F10 from popping up the application menu, open terminal and run:
     + $ gconftool-2 -s --type string /desktop/gnome/interface/menubar\_accel 'F12'

Basically the command above assigns menubar acceleration key to something other than F10, in this case F12 key

* + Mac:
    - System Preference --> Keyboard & Mouse --> Keyboard shortcuts
    - Uncheck F10

## Importing PJSIP

To import PJSIP into your Eclipse workspace:

1. Create a new project for PJPROJECT:
   1. Select **File** --> **New** --> **C Project** menu.
   2. In the C Project dialog:
      * Project name: **pjproject**
      * Uncheck Use default location checkbox
      * Location: set to the location of you PJPROJECT installation
      * Project type: **Makefile project** --> **Empty Project**
      * Toolchains: **Linux GCC**
   3. Click **Finish**
2. You should now see your project in the Project Explorer pane.

## Customizing PJSIP

1. Compile time customizations can be put in **user.mak** file in your root PJPROJECT directory. Example of a user.mak file content:
2. export CFLAGS += -g
3. export LDFLAGS +=
4. Other customizations can be put in **config\_site.h** as usual.

## Debugging the Project

1. Don't forget to put **-g** flag in your **user.mak** and possibly remove any optimization flags.
2. Don't forget to run **make dep** once, otherwise dependencies wouldn't have been set correctly.
3. Build the project
4. Create a new Debug configuration for each executable that you want to debug:
   1. Open **Run** --> **Debug Configurations**
   2. Select **C/C++ Application** from the left tree pane, and press **New** button
   3. Edit the Debug Configuration:
      1. Name: **pjsua** (for example)
      2. For the **C/C++ Application**, click **Search Project** and select the pjsua executable from there (that's why you need to build the project first)
      3. Go to Debugger tab:
         * if Debugger is not set, select **gdb/mi**
         * uncheck **Stop on startup at..**
      4. Click Apply
   4. Click **Debug** to start debugging
5. You may set breakpoints, inspect variables, step into/over, etc. as you would expect from a decent IDE.

## Problems and Solutions

### I have not been able to set breakpoints

Make sure the project is built with debugging flag (i.e. "-g")

### When I step into/over the code, the execution moves erraticly

Make sure optimization flag is turned off.

-----------------------------------------------------------------------------------------------------------------------------------

# Getting the Source Code Distribution

1. [Getting the Release tarball](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#GettingtheReleasetarball)
2. [Getting from Subversion trunk](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#GettingfromSubversiontrunk)
3. [Source Directories Layout](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#SourceDirectoriesLayout)
   1. [Top-Level Directory Layout](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#Top-LevelDirectoryLayout)
   2. [Individual Directory Inside Each Project](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#IndividualDirectoryInsideEachProject)

All libraries (PJLIB, PJLIB-UTIL, PJSIP, PJMEDIA, and PJMEDIA-CODEC) are currently distributed under a single source tree, collectively named as PJPROJECT or just PJ libraries. These libraries can be obtained by either downloading the release tarball or getting them from the source repository, the "Subversion trunk".

## Getting the Release tarball

Getting the released tarball, in ZIP or TGZ format, which is a certain snapshot in time.

The latest released tarball can always be downloaded from the [​http://www.pjsip.org/download.htm](http://www.pjsip.org/download.htm).

## Getting from Subversion trunk

PJPROJECT Subversion repository will always contain the latest/most up-to-date version of the sources. Normally the Subversion repository is always kept in a "good" state. However, there's always a chance that things break and the tree doesn't build correctly (particularly for the "not-so-popular" targets), so please consult the mailing list should there be any problems.

Using Subversion also has benefits of keeping the local copy of the source up to date with the main PJ source tree and to easily track the changes made to the local copy, if any. [Learn more about using Subversion](http://trac.pjsip.org/repos/wiki/Getting-Started/Using-Subversion).

The path to the repository is:

* For the current development version (2.x onwards):

[**​http://svn.pjsip.org/repos/pjproject/trunk**](http://svn.pjsip.org/repos/pjproject/trunk)

* For version 1.x:

[**​http://svn.pjsip.org/repos/pjproject/branches/1.x/**](http://svn.pjsip.org/repos/pjproject/branches/1.x/)

* For development projects not in any release, find it in:

[**​http://svn.pjsip.org/repos/pjproject/branches/projects**](http://svn.pjsip.org/repos/pjproject/branches/projects)

No userid or password is needed.

## Source Directories Layout

### Top-Level Directory Layout

The top-level directories (denoted as $TOP here) in the source distribution contains the following sub-directories:

$TOP/build

Contains makefiles that are common for all projects.

$TOP/build.symbian

Contains MMP files for building Symbian target.

$TOP/pjlib

Contains header and source files of PJLIB. PJLIB is the base portability and framework library which is used by all other libraries

$TOP/pjlib-util

Contains PJLIB-UTIL header and source files. PJLIB-UTIL is an auxiliary library that contains utility functions such as scanner, XML, STUN, MD5 algorithm, getopt() implementation, etc.

$TOP/pjnath

Contains PJNATH header and source files. PJNATH contains STUN, TURN, and ICE implementation.

$TOP/pjmedia

Contains PJMEDIA and PJMEDIA-CODEC header and source files. The sources of various codec **wrappers** (such as GSM, Speex, and iLBC) can be found under this directory. The codec libraries themselves normally will be put under $TOP/third\_party directory

$TOP/pjsip

Contains PJSIP header and source files. This library is the SIP protocol stack implementation.

$TOP/pjsip-apps

Contains source code for PJSUA and various sample applications, including the Python wrapper.

$TOP/tests

Contains various unit tests for the libraries.

$TOP/third\_party

Contains source code for various third party libraries, such as Speex, iLBC, and GSM codecs.

### Individual Directory Inside Each Project

Each library directory further contains these sub-directories:

bin

Contains binaries produced by the build process.

build

Contains build scripts/makefiles, project files, project workspace, etc. to build the project. In particular, it contains one Makefile file to build the project with GNU build systems, and a \*.dsw workspace file to build the library with Microsoft Visual Studio 6 or later.

build/output

The build/output directory contains the object files and other files generated by the build process. To support building multiple targets with a single source tree, each build target will occupy a different subdirectory under this directory.

docs

Contains Doxygen configuration file (doxygen.cfg) to generate online documentation from the source files. The output documentation will be put in this directory as well (for example, docs/html directory for the HTML files).

(to generate Doxygen documentation from the source tree, just run "doxygen docs/doxygen.cfg" in the individual project directory. The generated files will reside in docs directory).

include

Contains the header files for the project.

lib

Contains libraries produced by the build process.

src

Contains the source files of the project.

-----------------------------------------------------------------------------------------------------------------

# Build Preparation

1. [Create config\_site.h file](http://trac.pjsip.org/repos/wiki/Getting-Started/Build-Preparation#Createconfig_site.hfile)
2. [What is config\_site.h File](http://trac.pjsip.org/repos/wiki/Getting-Started/Build-Preparation#Whatisconfig_site.hFile)
3. [Creating config\_site.h file](http://trac.pjsip.org/repos/wiki/Getting-Started/Build-Preparation#Creatingconfig_site.hfile)

## Create config\_site.h file

Before source files can be built, the pjlib/include/pj/config\_site.h file must be created (it can just be an empty file).

Note: When the Makefile based build system is used, this process is taken care by the Makefiles. But when non-Makefile based build system (such as Visual Studio) is used, the config\_site.h file must be created manually.

## What is config\_site.h File

The pjlib/include/pj/config\_site.h contains local customizations to the libraries.

All customizations should be put in this file instead of modifying PJ's files, because if PJ's files get modified, then those modified files will not be updated the next time the source is synchronized. Or in other case, the local modification may be overwritten with the fresh copy from the SVN.

Putting the local customization to the config\_site.h solves this problem, because this file is not included in the version control, so it will never be overwritten by "svn update" command.

Please find list of configuration macros that can be overriden from these files:

* PJLIB Configuration (the pjlib/config.h file)
* PJLIB-UTIL Configuration (the pjlib-util/config.h file)
* PJNATH Configuration (the pjnath/config.h file)
* PJMEDIA Configuration (the pjmedia/config.h file)
* PJSIP Configuration (the pjsip/sip\_config.h file)

A sample config\_site.h file is also available in pjlib/include/config\_site\_sample.h.

## Creating config\_site.h file

The simplest way is just to create an empty file, to use whetever default values set by the libraries.

Another way to create the config\_site.h file is to write something like the following:

// Uncomment to get minimum footprint (suitable for 1-2 concurrent calls only)

//#define PJ\_CONFIG\_MINIMAL\_SIZE

// Uncomment to get maximum performance

//#define PJ\_CONFIG\_MAXIMUM\_SPEED

#include <pj/config\_site\_sample.h>

--------------------------------------------------------------------------------------------------------------------------------

# Getting Started: Building for Android

1. [Requirements](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#Requirements)
2. [Build Preparation](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#BuildPreparation)
3. [Building PJSIP](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#BuildingPJSIP)
4. [Building and running pjsua sample application](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#pjsua)
5. [Debugging native code with Eclipse](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#debug-native)
6. [Other Android projects](http://trac.pjsip.org/repos/wiki/Getting-Started/Android#OtherAndroidprojects)

Android target is currently only available in [latest source code from repository](https://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#GettingfromSubversiontrunk). Follow the instructions for the trunk, the development branch has been merged and is not maintained.

## Requirements

* Besides the [​Android SDK](http://developer.android.com/sdk/index.html), you will also need the [​Android NDK](http://developer.android.com/tools/sdk/ndk/index.html).
* Optional if you want to build and and run the sample application PJSUA:
  + [​SWIG](http://www.swig.org/download.html) (minimum version 2.0.5)
  + Telnet application to interact with PJSUA command line. If you are not familiar with telnet, please find a tutorial suitable for your development platform.

## Build Preparation

1. [Get the source code from repository](http://trac.pjsip.org/repos/wiki/Getting-Started/Download-Source#GettingfromSubversiontrunk), if you haven't already. This tutorial applies to PJSIP version 2.2 and above (or the 2.1 from SVN trunk dated 2013/04/24 or later).
2. Set your [config\_site.h](http://trac.pjsip.org/repos/wiki/Getting-Started/Build-Preparation) to the following:
3. #define PJ\_CONFIG\_ANDROID 1
4. #include <pj/config\_site\_sample.h>

This will activate Android specific settings in the config\_site\_sample.h.

## Building PJSIP

Just run:

$ cd /path/to/your/pjsip/dir

$ export ANDROID\_NDK\_ROOT=/path\_to\_android\_ndk\_dir

$ ./configure-android

$ make dep && make clean && make

Notes:

* to build for other targets, e.g: x86 or armv7, instead of just './configure-android', run
* TARGET\_ABI=armeabi-v7a ./configure-android --use-ndk-cflags
* for some targets, there may be error like
* error: undefined reference to '\_\_stack\_chk\_fail\_local'

this can be fixed by adding -fno-stack-protector into CFLAGS, e.g: via user.mak file.

* the ./configure-android is a wrapper that calls the standard ./configure script with settings suitable for Android target.
* you may pass standard ./configure options to this script too.
* for more info, run ./configure-android --help
* other customizations are similar to what is explained in [Building with GNU](http://trac.pjsip.org/repos/wiki/Getting-Started/Autoconf) page.

## Building and running pjsua sample application

We have pjsua sample application located under pjsip-apps/src/pjsua/android. It is not built by default, and you need [​SWIG](http://www.swig.org/download.html) to build it.

Follow these steps to build pjsua:

1. Make sure SWIG is in the build environment PATH.
   * Alternatively, update SWIG path in $PJDIR/pjsip-apps/src/pjsua/android/jni/Android.mk file.
2. Run ndk-build from directory $PJDIR/pjsip-apps/src/pjsua/android, note that the Android NDK root should be in the PATH, e.g:
3. $ cd /path/to/your/pjsip/dir
4. $ cd pjsip-apps/src/pjsua/android
5. $ ndk-build
6. Create Android project from pjsua. In Eclipse:
   * From menu: **File** --> **New** --> **Project**
   * Select **Android Project from Existing Code**, press **Next**
   * In Root Directory, put the location of **pjsua** source code (i.e. $PJDIR/pjsip-apps/src/pjsua/android) and press **Finish**
7. You may need to select different Android SDK than what is configured in pjsua. You can do this from the project's **Properties**.
8. Run it.
9. You will see telnet instructions on the device's screen. Telnet to this address to operate the application. See [PJSUA CLI Manual](http://trac.pjsip.org/repos/wiki/PJSUA-CLI) for command reference.

## Debugging native code with Eclipse

Here are the steps for debugging PJSIP native code using Eclipse:

1. Build PJSIP with debugging enabled, e.g: insert CFLAGS += -g into user.mak in PJSIP root directory.
2. Make sure that the JNI part of the application is built using ndk-build. For reference, check pjsua's Android.mk build config in pjsip-apps/src/pjsua/android/jni, it contains sample of how to import PJSIP build settings (build search paths, build flags, etc) and SWIG invocation.
3. Enable NDK plugin for Eclipse, check [​this](http://tools.android.com/recent/usingthendkplugin) and follow the instructions.
   * CDT can also be fetched from [​CDT download page](http://www.eclipse.org/cdt/downloads.php), normally Eclipse for C/C++ will have CDT installed, just make sure that the CDT version is 7.0.2 or newer.
4. It is recommended to introduce delay (about 5 seconds) in the application code between loading the native code library and calling any native functions (to be debugged), e.g:
5. try {
6. System.loadLibrary("some\_native\_lib.so");
7. } catch (UnsatisfiedLinkError e) {
8. return -1;
9. }
10. // Wait for GDB init
11. if ((getApplicationInfo().flags & ApplicationInfo.FLAG\_DEBUGGABLE) != 0) {
12. try {
13. Thread.sleep(5000);
14. } catch (InterruptedException e) { }
15. }
16. // Start calling native functions here
17. // ...
18. Load the PJSIP project to Eclipse (if not yet), and try put breakpoint anywhere in the PJSIP code before launching the Android application debug configuration in Eclipse.

## Other Android projects

Also have a look at the following PJSIP Android project:

* [​csipsimple](http://code.google.com/p/csipsimple/) project, an Android port of pjsip.

--------------------------

--------------------------------------------------------------------------------------------------------------------------------

# Running the Bundled Applications

1. [pjsua](http://trac.pjsip.org/repos/wiki/Getting-Started/Running-Applications#pjsua)
2. [Sample Applications](http://trac.pjsip.org/repos/wiki/Getting-Started/Running-Applications#SampleApplications)
3. [pjlib-test](http://trac.pjsip.org/repos/wiki/Getting-Started/Running-Applications#pjlib-test)
4. [pjsip-test](http://trac.pjsip.org/repos/wiki/Getting-Started/Running-Applications#pjsip-test)

Upon successful build, the output libraries (PJLIB, PJLIB-UTIL, PJMEDIA, PJSIP, etc.) are put under ./lib sub-directory under each project directory. In addition, some applications may also be built, and such applications will be put in ./bin sub-directory under each project directory.

## pjsua

pjsua is the reference implementation for both PJSIP and PJMEDIA stack, and is the main target of the build system. Upon successful build, pjsua application will be put in pjsip-apps/bin directory.

pjsua manual can be found in pjsua Manual Page.

## Sample Applications

Sample applications will be built with the Makefile build system. For Visual Studio, you have to build the samples manually by selecting and building the Samples project inside pjsip-apps/build/pjsip\_apps.dsw project workspace.

Upon successful build, the sample applications are put in pjsip-apps/bin/samples directory.

The sample applications are described in PJMEDIA Samples Page and PJSIP Samples Page in the website.

## pjlib-test

pjlib-test contains comprehensive tests for testing PJLIB functionality. This application will only be built when the Makefile build system is used; with Visual Studio, one has to open pjlib.dsw project in pjlib/build directory to build this application.

If you're porting PJLIB to new target, it is recommended to run this application to make sure that all functionalities works as expected.

## pjsip-test

pjsip-test contains codes for testing various SIP functionalities in PJSIP and also to benchmark static performance metrics such as message parsing per second.

------------------------------------------------------------------------------------------

# Building Application using PJSIP with GNU Tools

1. [Requirements](http://trac.pjsip.org/repos/wiki/Getting_Started_Using#Requirements)
2. [Steps for Building Your Application that Uses PJSIP/PJMEDIA](http://trac.pjsip.org/repos/wiki/Getting_Started_Using#StepsforBuildingYourApplicationthatUsesPJSIPPJMEDIA)
3. [Having Any Problems?](http://trac.pjsip.org/repos/wiki/Getting_Started_Using#HavingAnyProblems)
4. [Credits](http://trac.pjsip.org/repos/wiki/Getting_Started_Using#Credits)

## Requirements

* GNU tools (GNU make, binutils, gcc, and the likes).

## Steps for Building Your Application that Uses PJSIP/PJMEDIA

1. First, build pjproject libraries as described in [building for GNU systems](http://trac.pjsip.org/repos/wiki/Getting-Started/Autoconf). This normally is accomplished by executing these commands: $ ./configure && make dep && make
2. Create a directory outside the PJSIP sources for your project and place your source files there.
3. Create a file named **Makefile** in your source directory:
   1. If you have PJ **version 1.6** or later, **and** you run make install, **and** you have **pkg-config** tool, you can use this template for your Makefile:
   2. # If your application is in a file named myapp.cpp or myapp.c
   3. # this is the line you will need to build the binary.
   4. all: myapp
   5. myapp: myapp.cpp
   6. $(CC) -o $@ $< `pkg-config --cflags --libs libpjproject`
   7. clean:
   8. rm -f myapp.o myapp
   9. Otherwise if you have PJ **version 0.5.10.2** or later, you can use this template for your Makefile:
   10. #Modify this to point to the PJSIP location.
   11. PJBASE=/home/myself/pjproject-0.5.10.2
   12. include $(PJBASE)/build.mak
   13. CC = $(PJ\_CC)
   14. LDFLAGS = $(PJ\_LDFLAGS)
   15. LDLIBS = $(PJ\_LDLIBS)
   16. CFLAGS = $(PJ\_CFLAGS)
   17. CPPFLAGS= ${CFLAGS}
   18. # If your application is in a file named myapp.cpp or myapp.c
   19. # this is the line you will need to build the binary.
   20. all: myapp
   21. myapp: myapp.cpp
   22. $(CC) -o $@ $< $(CPPFLAGS) $(LDFLAGS) $(LDLIBS)
   23. clean:
   24. rm -f myapp.o myapp
   25. Otherwise if you have PJ **version 0.5.10.1** or older, you can use this template for your Makefile:
   26. # Modify this to point to the PJSIP location.
   27. PJBASE=/home/myself/pjproject-0.5.10.1
   28. include $(PJBASE)/build/mak
   29. CC=$(CROSS\_COMPILE)$(CC\_NAME)
   30. # Remove components that you don't need from the following definitions.
   31. LDFLAGS=-L${PJBASE}/pjlib/lib\
   32. -L${PJBASE}/pjlib-util/lib\
   33. -L${PJBASE}/pjmedia/lib\
   34. -L${PJBASE}/pjsip/lib
   35. LDLIBS=-lpjsua-${TARGET\_NAME}\
   36. -lpjsip-ua-${TARGET\_NAME}\
   37. -lpjsip-simple-${TARGET\_NAME}\
   38. -lpjsip-${TARGET\_NAME}\
   39. -lpjmedia-codec-${TARGET\_NAME}\
   40. -lpjmedia-${TARGET\_NAME}\
   41. -lpjmedia-codec-${TARGET\_NAME}\
   42. -lpjlib-util-${TARGET\_NAME}\
   43. -lpj-${TARGET\_NAME}\
   44. -lm\
   45. -lpthread\
   46. -lasound\
   47. -lssl
   48. CFLAGS=-I${PJBASE}/pjlib/include\
   49. -I${PJBASE}/pjlib-util/include\
   50. -I${PJBASE}/pjmedia/include\
   51. -I${PJBASE}/pjsip/include\
   52. -DPJ\_AUTOCONF=1
   53. CPPFLAGS=${CFLAGS}
   54. # If your application is in a file named myapp.cpp or myapp.c
   55. # this is the line you will need to build the binary.
   56. all: myapp
   57. myapp: myapp.cpp
   58. $(CC) -o $@ $< $(CPPFLAGS) $(LDFLAGS) $(LDLIBS)
   59. clean:
   60. rm -f myapp.o myapp
4. There few things to note when making the **Makefile** above:
   1. First, make sure that you replace **PJBASE** with the location of PJSIP sources in your computer.
   2. If you notice there are spaces towards the bottom of the file (before $(CC) and rm, these are a single tab, not spaces. **This is important**, or otherwise **make** command will fail with "**missing separator**" error.
   3. Change myapp.cpp to your source filename.
   4. If you're using version 0.5.10.1 or older, then you may encounter link error with **-lasound** or **-lssl**. You may remove this from your Makefile (this is the limitation of 0.5.10.1 build system, which has been fixed in 0.5.10.2).
5. Create myapp.cpp in the same directory as your Makefile. At minimum, it may look like this:
6. #include <pjlib.h>
7. #include <pjlib-util.h>
8. #include <pjmedia.h>
9. #include <pjmedia-codec.h>
10. #include <pjsip.h>
11. #include <pjsip\_simple.h>
12. #include <pjsip\_ua.h>
13. #include <pjsua-lib/pjsua.h>
14. int main()
15. {
16. return 0;
17. }
18. Last, run **make** in your source directory.

## Having Any Problems?

If you're having any problems or encounter some errors with above instructions, first please watch carefully the error messages that are printed to the screen. The error messages printed by the gcc or **make** normally should contain useful information about the error and how to fix it.

If you like more help, you can report the error to PJSIP mailing list. Don't forget to include the details of the error (such as copy/paste the compiler command and error output), because without this we simply wouldn't know how to help!

## Credits

Thanks Binu KS <binuks -at- gmail dot com> for the initial tutorial in the PJSIP mailing list.

------------------------------------------------

# PJSIP Video User's Guide

**Table of Contents**

1. [Building with Video Support](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#BuildingwithVideoSupport)
2. [Building the GUI Sample Application](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vidgui)
   1. [GNU Build System (Mac OS X, Linux, etc)](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#GNUBuildSystemMacOSXLinuxetc)
   2. [Visual Studio](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#VisualStudio)
3. [Using Video API (pjsua-lib)](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#UsingVideoAPIpjsua-lib)
   1. [Enabling Video](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#EnablingVideo)
   2. [Incoming Video Call](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#IncomingVideoCall)
      1. [Display Incoming Video Automatically](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#DisplayIncomingVideoAutomatically)
      2. [Show or Hide Incoming Video](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#ShoworHideIncomingVideo)
      3. [Controlling Incoming Video Stream](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#civs)
      4. [Incoming Re-offer](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#IncomingRe-offer)
   3. [Outgoing Video Call](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#OutgoingVideoCall)
      1. [Outgoing Video Transmission](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#OutgoingVideoTransmission)
      2. [Default Capture Device](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#DefaultCaptureDevice)
      3. [Controlling Video Stream](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#cvs)
   4. [Add or Remove Video](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#AddorRemoveVideo)
   5. [Working with Video Window](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#wwin)
   6. [Modifying video codec parameters for video call](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#modvcodec)
      1. [Size or resolution](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#Sizeorresolution)
      2. [Framerate](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#Framerate)
      3. [Bitrate](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#Bitrate)
4. [Additional Info](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#AdditionalInfo)
   1. [Using OpenGL with SDL](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#sdl-opengl)
5. [Video API Reference (pjsua-lib)](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vidref)
   1. [Device enumeration API](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#DeviceenumerationAPI)
   2. [Video preview API](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#VideopreviewAPI)
   3. [Video Configuration](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vconf)
   4. [Video Call Manipulation](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vcm)
   5. [Video Call Information](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vci)
   6. [Video Call Stream Information and Statistic](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vcsi)
   7. [Video Window API](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vw)
   8. [Video Codec API](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vc)

Video is available on PJSIP version 2.0 and later. Only desktop platforms are supported, mobile devices such as iOS are not yet supported. This document describes how to use the video feature with PJSIP.

## Building with Video Support

Follow [Getting Started](http://trac.pjsip.org/repos/wiki/Getting-Started) for your platform on building pjsip with video support.

## Building the GUI Sample Application

We have a GUI sample application with video support. The project is located under pjsip-apps/src/vidgui. It is not built by default, and you need [​Qt SDK](http://qt.nokia.com/downloads/) to build it.

### GNU Build System (Mac OS X, Linux, etc)

Follow these steps to build vidgui sample:

1. Go to vidgui source directory:
2. $ cd pjsip-apps/src/vidgui
3. Generate Makefile. For Linux:
4. $ qmake

and for Mac OS X:

$ qmake -spec macx-g++

1. Build the app:
2. $ make

### Visual Studio

Follow these steps to build vidgui sample with Visual Studio:

1. Open command prompt, and
2. cd pjsip-apps\src\vidgui
3. Generate project files:
4. qmake -tp vc
5. Open vidgui.vcproj project.
6. Save the solution, and build the project

## Using Video API (pjsua-lib)

This section provides several sample scenarios of using video in your application. Please see [Video API Reference](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vidref) section for a more complete documentation about the Video API.

### Enabling Video

By default, video is enabled in **pjsua\_call\_setting**, via **vid\_cnt** setting.

### Incoming Video Call

Incoming video will be accepted/rejected depending on whether video is enabled in the call setting (see above). You can pass the call setting using the API pjsua\_call\_answer2() (so for example, to reject the video, set **vid\_cnt** to 0 and call pjsua\_call\_answer2()). If video is enabled, incoming video will be accepted as long as we have matching codec for it. However, this does not necessarily mean that the video will be displayed automatically to the screen, nor that outgoing video will be transmitted automatically, as there will be separate settings for these. Outgoing video behavior will be explained in the next section.

#### Display Incoming Video Automatically

By default, incoming video **is not** displayed automatically, since the app may want to seek user approval first. Use the following code to change this behavior on per account basis:

pjsua\_acc\_config cfg;

pjsua\_acc\_config\_default(&cfg);

cfg.vid\_in\_auto\_show = PJ\_TRUE;

#### Show or Hide Incoming Video

Regardless of the setting above, you can use the following steps to show or hide the display incoming video:

1. Use pjsua\_call\_get\_vid\_stream\_idx() or enumerate the call's media stream to find the media index of the default video. If there are multiple video streams in a call, the default video is the first active video media in the call.
2. Locate the media information of the specified stream index in the pjsua\_call\_info, and acquire the window ID associated with the remote video. Sample code:
3. int vid\_idx;
4. pjsua\_vid\_win\_id wid;
5. vid\_idx = pjsua\_call\_get\_vid\_stream\_idx(call\_id);
6. if (vid\_idx >= 0) {
7. pjsua\_call\_info ci;
8. pjsua\_call\_get\_info(call\_id, &ci);
9. wid = ci.media[vid\_idx].stream.vid.win\_in;
10. }
11. Using the video window ID, you may retrieve the associated native video handle with pjsua\_vid\_win\_get\_info() and then show or hide the video window using native API, or use pjsua\_vid\_win\_set\_show() to show/hide the window using PJSUA API. See [Working with Window](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#wwin) section below for information on manipulating video windows.

#### Controlling Incoming Video Stream

Controlling the video window above will not cause any re-INVITE or UPDATE to be sent to remote, since the operation occurs locally. However, if you wish, you may alter the incoming video stream with pjsua\_call\_set\_vid\_strm() API, and this **will** cause re-INVITE or UPDATE to be sent to negotiate the new SDP. The relevant operation to control incoming video with pjsua\_call\_set\_vid\_strm() are:

* PJSUA\_CALL\_VID\_STRM\_CHANGE\_DIR: change the media direction (e.g. to "sendonly", or even "inactive")
* PJSUA\_CALL\_VID\_STRM\_REMOVE: remove the media stream altogether by settings its port to zero
* PJSUA\_CALL\_VID\_STRM\_ADD: add new video media stream

Since pjsua\_call\_set\_vid\_strm() will result in renegotiation of the SDP in a re-INVITE or UPDATE transaction, the result of this operation will not be available immediately. Application can monitor the status by implementing on\_call\_media\_state() callback and enumerate the media stream status with pjsua\_call\_info.

#### Incoming Re-offer

If the re-offer contains video, incoming re-offer will be automatically answered with current video setting in the call setting. Currently there is no callback for this, however application can always watch for media update via on\_call\_media\_state() callback.

### Outgoing Video Call

Outgoing video is enabled/disabled depending on the call setting. To initiate a call with video in the SDP as inactive, you can disable the video in the call setting and set **pjsua\_call\_setting.flag** with **PJSUA\_CALL\_INCLUDE\_DISABLED\_MEDIA**.

#### Outgoing Video Transmission

Outgoing video transmission is independent from the incoming video transmission; each can be operated separately. Note that outgoing video transmission **is not started by default**, not even when incoming offer contains video support. This behavior is controlled by pjsua\_acc\_config.vid\_out\_auto\_transmit setting, which default to PJ\_FALSE. Setting this to PJ\_TRUE will cause video transmission to be started automatically on each outgoing calls and on incoming calls that indicates video support in its offer. However, it is more flexible and appropriate to leave this setting at PJ\_FALSE, and add video later during the call by using pjsua\_call\_set\_vid\_strm() API, as will be explained shortly.

#### Default Capture Device

The default capture device that is used by an account is configured in pjsua\_acc\_config.vid\_cap\_dev setting. It is more convenient to set the "correct" device here rather than having to set it in every other API calls later.

#### Controlling Video Stream

Application uses pjsua\_call\_set\_vid\_strm() API to control video stream on a call. The following op\_code can be used on the API to control the video media:

* PJSUA\_CALL\_VID\_STRM\_ADD: add a new video stream
* PJSUA\_CALL\_VID\_STRM\_REMOVE: remove video stream (set port to zero)
* PJSUA\_CALL\_VID\_STRM\_CHANGE\_DIR: change direction or deactivate (i.e. set direction to "inactive")
* PJSUA\_CALL\_VID\_STRM\_CHANGE\_CAP\_DEV: change capture device
* PJSUA\_CALL\_VID\_STRM\_START\_TRANSMIT: start previously stopped transmission
* PJSUA\_CALL\_VID\_STRM\_STOP\_TRANSMIT: stop transmission

Some of the video operations above require re-INVITE or UPDATE to be sent, hence the result will not be available immediately. In that case, application can implement on\_call\_media\_state() callback and inspect the resulting negotiation by looking at the pjsua\_call\_info. Please see [Video Call Manipulation](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#vcm) in the API reference section below for more information about the operations above.

### Add or Remove Video

You can set the field **vid\_cnt** of **pjsua\_call\_setting** to the desired video count to add/remove video, then send the reinvite/update. Alternatively, you can use pjsua\_call\_set\_vid\_strm() API to control the video stream on a call (see [Controlling Incoming Video Stream](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#civs) or [Controlling Video Stream](http://trac.pjsip.org/repos/wiki/Video_Users_Guide#cvs) above).

### Working with Video Window

Video Window represents all window objects on the screen that the library creates. The video window can display incoming video, preview, and/or other video playbacks.

Application may retrieve video windows from the following places:

* for calls, the video window of incoming video stream is contained in the media stream inside pjsua\_call\_info structure.
* preview window associated with a capture device can be queried with pjsua\_vid\_preview\_get\_win().
* for all other purposes, application may enumerate all video windows with pjsua\_vid\_enum\_wins().

Application retrieves pjsua\_vid\_win\_info with pjsua\_vid\_win\_get\_info(). The one window property that most applications will be interested with is the native window handle of the video. The native video handle is contained by pjmedia\_vid\_dev\_hwnd structure inside pjsua\_vid\_win\_info. Application can use the native handle to embed the video window into application's GUI structure. Alternatively, the library also provides few simple and most commonly used API to operate the window, such as pjsua\_vid\_win\_set\_show(), pjsua\_vid\_win\_set\_size(), etc., however the availability of these APIs are not guaranteed since it depends on the underlying backend device.

### Modifying video codec parameters for video call

Video codec parameters are specified in pjmedia\_vid\_codec\_param. The codec parameters provide separate settings for each direction, encoding and decoding. Any modifications on video codec parameters can be applied using pjsua\_vid\_codec\_set\_param(), here is a sample code for reference:

const pj\_str\_t codec\_id = {"H264", 4};

pjmedia\_vid\_codec\_param param;

pjsua\_vid\_codec\_get\_param(&codec\_id, &param);

/\* Modify param here \*/

...

pjsua\_vid\_codec\_set\_param(&codec\_id, &param);

#### Size or resolution

Specify video picture dimension.

1. For encoding direction, configured via pjmedia\_vid\_codec\_param.enc\_fmt.det.vid.size, e.g:
2. /\* Sending 1280 x 720 \*/
3. param.enc\_fmt.det.vid.size.w = 1280;
4. param.enc\_fmt.det.vid.size.h = 720;

Note that there is a possibility that the value will be adjusted to follow remote capability. For example, if remote signals that maximum resolution supported is 640 x 480 and locally the encoding direction size is set to 1280 x 720, then 640 x 480 will be used.

1. For decoding direction, two steps are needed:
   1. pjmedia\_vid\_codec\_param.dec\_fmt.det.vid.size should be set to the highest value expected for incoming video size.
   2. signalling to remote, configured via codec specific SDP format parameter (fmtp): pjmedia\_vid\_codec\_param.dec\_fmtp.
      * H263-1998, e.g:
      * /\* 1st preference: 352 x 288 (CIF) \*/
      * param.dec\_fmtp.param[n].name = pj\_str("CIF");
      * /\* The value actually specifies framerate, see framerate section below \*/
      * param.dec\_fmtp.param[n].val = pj\_str("1");
      * /\* 2nd preference: 176 x 144 (QCIF) \*/
      * param.dec\_fmtp.param[n+1].name = pj\_str("QCIF");
      * /\* The value actually specifies framerate, see framerate section below \*/
      * param.dec\_fmtp.param[n+1].val = pj\_str("1");
      * H264, the size is implicitly specified in H264 level (check the standard specification or [​this](http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC#Levels)) and on SDP, the H264 level is signalled via H264 SDP fmtp [​profile-level-id](http://tools.ietf.org/html/rfc6184#section-8.1), e.g:
      * /\* Can receive up to 1280×720 @30fps \*/
      * param.dec\_fmtp.param[n].name = pj\_str("profile-level-id");
      * /\* Set the profile level to "1f", which means level 3.1 \*/
      * param.dec\_fmtp.param[n].val = pj\_str("xxxx1f");

#### Framerate

Specify number of frames processed per second.

1. For encoding direction, configured via pjmedia\_vid\_codec\_param.enc\_fmt.det.vid.fps, e.g:
2. /\* Sending @30fps \*/
3. param.enc\_fmt.det.vid.fps.num = 30;
4. param.enc\_fmt.det.vid.fps.denum = 1;

Note:

* + that there is a possibility that the value will be adjusted to follow remote capability. For example, if remote signals that maximum framerate supported is 10fps and locally the encoding direction framerate is set to 30fps, then 10fps will be used.
  + **limitation:** if preview is enabled before call is established, capture device will opened using default framerate of the device, and subsequent calls that use that device will use this framerate regardless of the configured encoding framerate that is set above. Currently the only solution is to disable preview before establishing media and re-enable it once the video media is established.

1. For decoding direction, two steps are needed:
   * pjmedia\_vid\_codec\_param.dec\_fmt.det.vid.fps should be set to the highest value expected for incoming video framerate.
   * signalling to remote, configured via codec specific SDP format parameter (fmtp): pjmedia\_vid\_codec\_param.dec\_fmtp.
     + H263-1998, maximum framerate is specified per size/resolution basis, check [​here](http://tools.ietf.org/html/rfc4629#section-8.1.1) for more info.
     + /\* 3000/(1.001\*2) fps for CIF \*/
     + param.dec\_fmtp.param[m].name = pj\_str("CIF");
     + param.dec\_fmtp.param[m].val = pj\_str("2");
     + /\* 3000/(1.001\*1) fps for QCIF \*/
     + param.dec\_fmtp.param[n].name = pj\_str("QCIF");
     + param.dec\_fmtp.param[n].val = pj\_str("1");
     + H264, similar to size/resolution, the framerate is implicitly specified in H264 level (check the standard specification or [​this](http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC#Levels)) and the H264 level is signalled via H264 SDP fmtp profile-level-id, e.g:
     + /\* Can receive up to 1280×720 @30fps \*/
     + param.dec\_fmtp.param[n].name = pj\_str("profile-level-id");
     + param.dec\_fmtp.param[n].val = pj\_str("xxxx1f");

#### Bitrate

Specify bandwidth requirement for video payloads stream delivery.

This is configurable via pjmedia\_vid\_codec\_param.enc\_fmt.det.vid.avg\_bps and pjmedia\_vid\_codec\_param.enc\_fmt.det.vid.max\_bps, e.g:

/\* Bitrate range preferred: 512-1024kbps \*/

param.enc\_fmt.det.vid.avg\_bps = 512000;

param.enc\_fmt.det.vid.max\_bps = 1024000;

Notes:

* This setting is applicable for encoding and decoding direction, currently there is no way to set asymmetric bitrate. By decoding direction, actually it just means that this setting will be queried when generating bandwidth info for local SDP (see next point).
* The bitrate setting of all codecs will be enumerated and the highest value will be signalled in bandwidth info in local SDP (see ticket [#1244](http://trac.pjsip.org/repos/ticket/1244)).
* There is a possibility that the encoding bitrate will be adjusted to follow remote bitrate setting, i.e: read from SDP bandwidth info (b=TIAS line) in remote SDP. For example, if remote signals that maximum bitrate is 128kbps and locally the bitrate is set to 512kbps, then 128kbps will be used.
* If codec specific bitrate setting signalling (via SDP fmtp) is desired, e.g: MaxBR for H263, application should put the SDP fmtp manually, for example:
* /\* H263 specific maximum bitrate 512kbps \*/
* param.dec\_fmtp.param[n].name = pj\_str("MaxBR");
* param.dec\_fmtp.param[n].val = pj\_str("5120"); /\* = max\_bps / 100 \*/

## Additional Info

### Using OpenGL with SDL

PJSIP supports OpenGL video rendering with SDL. Follow these steps to enable and use the OpenGL backend.

1. Install OpenGL development libraries for your system. The instructions vary, and some platforms may have OpenGL development libraries installed by default.
   * For Ubuntu 12.04, you can run the following:
   * $ sudo apt-get install freeglut3 freeglut3-dev
   * $ sudo apt-get install binutils-gold
2. Enable OpenGL support in SDL. With GNU tools, OpenGL support will be detected automatically by the "configure" script. Make sure that the "configure" output shows that OpenGL has been detected, by looking for line like this in the output:
3. checking for OpenGL (GLX) support... yes
4. Enable SDL OpenGL support in PJSIP, by declaring this in your config\_site.h:
5. #define PJMEDIA\_VIDEO\_DEV\_SDL\_HAS\_OPENGL 1
6. If you're not using Visual Studio, add OpenGL library in your application's input library list. If you're using GNU tools, you can add this in **user.mak** file in root PJSIP directory:
7. export LDFLAGS += -lGL
8. Rebuild PJSIP
9. Now **"SDL openGL renderer"** device should show up in video device list. Simply just use this device.

## Video API Reference (pjsua-lib)

This section explains and lists the Video API as it was available when this document is written. For a richer and more up to date list, please see [​Video API reference doxygen documentation](http://www.pjsip.org/docs/latest-2/pjsip/docs/html/group__PJSUA__LIB__VIDEO.htm).

The Video API is classified into the following categories.

### Device enumeration API

The following API is available:

/\*\*

\* Get the number of video devices installed in the system.

\*

\* @return The number of devices.

\*/

PJ\_DECL(unsigned) pjsua\_vid\_dev\_count(void);

/\*\*

\* Retrieve the video device info for the specified device index.

\*

\* @param id The device index.

\* @param vdi Device info to be initialized.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_dev\_get\_info(pjmedia\_vid\_dev\_index id,

pjmedia\_vid\_dev\_info \*vdi);

/\*\*

\* Enum all video devices installed in the system.

\*

\* @param info Array of info to be initialized.

\* @param count On input, specifies max elements in the array.

\* On return, it contains actual number of elements

\* that have been initialized.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_enum\_devs(pjmedia\_vid\_dev\_info info[],

unsigned \*count);

In addition, the [​PJMEDIA videodev](http://www.pjsip.org/docs/latest-2/pjmedia/docs/html/group__video__device__reference.htm) also provides this API to detect change in device availability:

/\*\*

\* Refresh the list of video devices installed in the system. This function

\* will only refresh the list of videoo device so all active video streams will

\* be unaffected. After refreshing the device list, application MUST make sure

\* to update all index references to video devices (i.e. all variables of type

\* pjmedia\_vid\_dev\_index) before calling any function that accepts video device

\* index as its parameter.

\*

\* @return PJ\_SUCCESS on successful operation or the appropriate

\* error code.

\*/

PJ\_DECL(pj\_status\_t) pjmedia\_vid\_dev\_refresh(void);

### Video preview API

The video preview API can be used to show the output of capture device to a video window:

/\*\*

\* Parameters for starting video preview with pjsua\_vid\_preview\_start().

\* Application should initialize this structure with

\* pjsua\_vid\_preview\_param\_default().

\*/

typedef struct pjsua\_vid\_preview\_param

{

/\*\*

\* Device ID for the video renderer to be used for rendering the

\* capture stream for preview.

\*/

pjmedia\_vid\_dev\_index rend\_id;

} pjsua\_vid\_preview\_param;

/\*\*

\* Start video preview window for the specified capture device.

\*

\* @param id The capture device ID where its preview will be

\* started.

\* @param prm Optional video preview parameters. Specify NULL

\* to use default values.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_preview\_start(pjmedia\_vid\_dev\_index id,

pjsua\_vid\_preview\_param \*prm);

/\*\*

\* Get the preview window handle associated with the capture device, if any.

\*

\* @param id The capture device ID.

\*

\* @return The window ID of the preview window for the

\* specified capture device ID, or NULL if preview

\* does not exist.

\*/

PJ\_DECL(pjsua\_vid\_win\_id) pjsua\_vid\_preview\_get\_win(pjmedia\_vid\_dev\_index id);

/\*\*

\* Stop video preview.

\*

\* @param id The capture device ID.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_preview\_stop(pjmedia\_vid\_dev\_index id);

### Video Configuration

Video is enabled/disabled on pjsua\_call\_setting:

/\*\*

\* Number of simultaneous active video streams for this call. Setting

\* this to zero will disable video in this call.

\*

\* Default: 1 (if video feature is enabled, otherwise it is zero)

\*/

unsigned vid\_cnt;

Video settings are mostly configured on pjsua\_acc\_config with the following fields:

/\*\*

\* Specify whether incoming video should be shown to screen by default.

\* This applies to incoming call (INVITE), incoming re-INVITE, and

\* incoming UPDATE requests.

\*

\* Regardless of this setting, application can detect incoming video

\* by implementing \a on\_call\_media\_state() callback and enumerating

\* the media stream(s) with #pjsua\_call\_get\_info(). Once incoming

\* video is recognised, application may retrieve the window associated

\* with the incoming video and show or hide it with

\* #pjsua\_vid\_win\_set\_show().

\*

\* Default: PJ\_FALSE

\*/

pj\_bool\_t vid\_in\_auto\_show;

/\*\*

\* Specify whether outgoing video should be activated by default when

\* making outgoing calls and/or when incoming video is detected. This

\* applies to incoming and outgoing calls, incoming re-INVITE, and

\* incoming UPDATE. If the setting is non-zero, outgoing video

\* transmission will be started as soon as response to these requests

\* is sent (or received).

\*

\* Regardless of the value of this setting, application can start and

\* stop outgoing video transmission with #pjsua\_call\_set\_vid\_strm().

\*

\* Default: PJ\_FALSE

\*/

pj\_bool\_t vid\_out\_auto\_transmit;

/\*\*

\* Specify the default capture device to be used by this account. If

\* \a vid\_out\_auto\_transmit is enabled, this device will be used for

\* capturing video.

\*

\* Default: PJMEDIA\_VID\_DEFAULT\_CAPTURE\_DEV

\*/

pjmedia\_vid\_dev\_index vid\_cap\_dev;

/\*\*

\* Specify the default rendering device to be used by this account.

\*

\* Default: PJMEDIA\_VID\_DEFAULT\_RENDER\_DEV

\*/

pjmedia\_vid\_dev\_index vid\_rend\_dev;

### Video Call Manipulation

The default video behavior for a call is controlled by the account settings above. On top of that, the application can manipulate video of an already-going call by using pjsua\_call\_set\_vid\_strm() API:

/\*\*

\* This enumeration represents video stream operation on a call.

\* See also #pjsua\_call\_vid\_strm\_op\_param for further info.

\*/

typedef enum pjsua\_call\_vid\_strm\_op

{

/\*\*

\* No operation

\*/

PJSUA\_CALL\_VID\_STRM\_NO\_OP,

/\*\*

\* Add a new video stream. This will add a new m=video line to

\* the media, regardless of whether existing video is/are present

\* or not. This will cause re-INVITE or UPDATE to be sent to remote

\* party.

\*/

PJSUA\_CALL\_VID\_STRM\_ADD,

/\*\*

\* Remove/disable an existing video stream. This will

\* cause re-INVITE or UPDATE to be sent to remote party.

\*/

PJSUA\_CALL\_VID\_STRM\_REMOVE,

/\*\*

\* Change direction of a video stream. This operation can be used

\* to activate or deactivate an existing video media. This will

\* cause re-INVITE or UPDATE to be sent to remote party.

\*/

PJSUA\_CALL\_VID\_STRM\_CHANGE\_DIR,

/\*\*

\* Change capture device of a video stream. This will not send

\* re-INVITE or UPDATE to remote party.

\*/

PJSUA\_CALL\_VID\_STRM\_CHANGE\_CAP\_DEV,

/\*\*

\* Start transmitting video stream. This will cause previously

\* stopped stream to start transmitting again. Note that no

\* re-INVITE/UPDATE is to be transmitted to remote since this

\* operation only operates on local stream.

\*/

PJSUA\_CALL\_VID\_STRM\_START\_TRANSMIT,

/\*\*

\* Stop transmitting video stream. This will cause the stream to

\* be paused in TX direction, causing it to stop sending any video

\* packets. No re-INVITE/UPDATE is to be transmitted to remote

\* with this operation.

\*/

PJSUA\_CALL\_VID\_STRM\_STOP\_TRANSMIT,

/\*\*

\* Send keyframe in the video stream. This will force the stream to

\* generate and send video keyframe as soon as possible. No

\* re-INVITE/UPDATE is to be transmitted to remote with this operation.

\*/

PJSUA\_CALL\_VID\_STRM\_SEND\_KEYFRAME

} pjsua\_call\_vid\_strm\_op;

/\*\*

\* Parameters for video stream operation on a call.

\*/

typedef struct pjsua\_call\_vid\_strm\_op\_param

{

/\*\*

\* Specify the media stream index. This can be set to -1 to denote

\* the default video stream in the call, which is the first active

\* video stream or any first video stream if none is active.

\*

\* This field is valid for all video stream operations, except

\* PJSUA\_CALL\_VID\_STRM\_ADD.

\*

\* Default: -1 (first active video stream, or any first video stream

\* if none is active)

\*/

int med\_idx;

/\*\*

\* Specify the media stream direction.

\*

\* This field is valid for the following video stream operations:

\* PJSUA\_CALL\_VID\_STRM\_ADD and PJSUA\_CALL\_VID\_STRM\_CHANGE\_DIR.

\*

\* Default: PJMEDIA\_DIR\_ENCODING\_DECODING

\*/

pjmedia\_dir dir;

/\*\*

\* Specify the video capture device ID. This can be set to

\* PJMEDIA\_VID\_DEFAULT\_CAPTURE\_DEV to specify the default capture

\* device as configured in the account.

\*

\* This field is valid for the following video stream operations:

\* PJSUA\_CALL\_VID\_STRM\_ADD and PJSUA\_CALL\_VID\_STRM\_CHANGE\_CAP\_DEV.

\*

\* Default: capture device configured in account.

\*/

pjmedia\_vid\_dev\_index cap\_dev;

} pjsua\_call\_vid\_strm\_op\_param;

/\*\*

\* Add, remove, modify, and/or manipulate video media stream for the

\* specified call. This may trigger a re-INVITE or UPDATE to be sent

\* for the call.

\*

\* @param call\_id Call identification.

\* @param op The video stream operation to be performed,

\* possible values are #pjsua\_call\_vid\_strm\_op.

\* @param param The parameters for the video stream operation,

\* or NULL for the default parameter values

\* (see #pjsua\_call\_vid\_strm\_op\_param).

\*

\* @return PJ\_SUCCESS on success or the appropriate error.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_call\_set\_vid\_strm (

pjsua\_call\_id call\_id,

pjsua\_call\_vid\_strm\_op op,

const pjsua\_call\_vid\_strm\_op\_param \*param);

/\*\*

\* Get the media stream index of the default video stream in the call.

\* Typically this will just retrieve the stream index of the first

\* activated video stream in the call.

\*

\* @param call\_id Call identification.

\*

\* @return The media stream index or -1 if no video stream

\* is present in the call.

\*/

PJ\_DECL(int) pjsua\_call\_get\_vid\_stream\_idx(pjsua\_call\_id call\_id);

### Video Call Information

New fields have been added to pjsua\_call\_info to report media information in more detail:

Snippet of pjsua\_call\_info new fields:

/\*\* Number of media streams in this call \*/

unsigned media\_cnt;

/\*\* Array of media stream information \*/

struct

{

/\*\* Media index in SDP. \*/

unsigned index;

/\*\* Media type. \*/

pjmedia\_type type;

/\*\* Media direction. \*/

pjmedia\_dir dir;

/\*\* Call media status. \*/

pjsua\_call\_media\_status status;

/\*\* The specific media stream info. \*/

union {

/\*\* Audio stream \*/

struct {

/\*\* The conference port number for the call. \*/

pjsua\_conf\_port\_id conf\_slot;

} aud;

/\*\* Video stream \*/

struct {

/\*\*

\* The window id for incoming video, if any, or

\* PJSUA\_INVALID\_ID.

\*/

pjsua\_vid\_win\_id win\_in;

/\*\* The video capture device for outgoing transmission,

\* if any, or PJMEDIA\_VID\_INVALID\_DEV

\*/

pjmedia\_vid\_dev\_index cap\_dev;

} vid;

} stream;

} media[PJMEDIA\_MAX\_SDP\_MEDIA];

### Video Call Stream Information and Statistic

New API has been added to query call's stream information and statistic. Subsequently, the old pjsua\_call\_get\_media\_session() API has been deprecated since its use is unsafe.

New API:

/\*\*

\* Media stream info.

\*/

typedef struct pjsua\_stream\_info

{

/\*\* Media type of this stream. \*/

pjmedia\_type type;

/\*\* Stream info (union). \*/

union {

/\*\* Audio stream info \*/

pjmedia\_stream\_info aud;

/\*\* Video stream info \*/

pjmedia\_vid\_stream\_info vid;

} info;

} pjsua\_stream\_info;

/\*\*

\* Media stream statistic.

\*/

typedef struct pjsua\_stream\_stat

{

/\*\* RTCP statistic. \*/

pjmedia\_rtcp\_stat rtcp;

/\*\* Jitter buffer statistic. \*/

pjmedia\_jb\_state jbuf;

} pjsua\_stream\_stat;

/\*\*

\* Get media stream info for the specified media index.

\*

\* @param call\_id The call identification.

\* @param med\_idx Media stream index.

\* @param psi To be filled with the stream info.

\*

\* @return PJ\_SUCCESS on success or the appropriate error.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_call\_get\_stream\_info(pjsua\_call\_id call\_id,

unsigned med\_idx,

pjsua\_stream\_info \*psi);

/\*\*

\* Get media stream statistic for the specified media index.

\*

\* @param call\_id The call identification.

\* @param med\_idx Media stream index.

\* @param psi To be filled with the stream statistic.

\*

\* @return PJ\_SUCCESS on success or the appropriate error.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_call\_get\_stream\_stat(pjsua\_call\_id call\_id,

unsigned med\_idx,

pjsua\_stream\_stat \*stat);

/\*\*

\* Get media transport info for the specified media index.

\*

\* @param call\_id The call identification.

\* @param med\_idx Media stream index.

\* @param t To be filled with the transport info.

\*

\* @return PJ\_SUCCESS on success or the appropriate error.

\*/

PJ\_DECL(pj\_status\_t)

pjsua\_call\_get\_med\_transport\_info(pjsua\_call\_id call\_id,

unsigned med\_idx,

pjmedia\_transport\_info \*t);

### Video Window API

A video window is a rectangular area in your monitor to display video content. The video content may come from remote stream, local camera (in case of preview), AVI playback, or any other video playback. Application mostly will be interested in the native handle of the video window so that it can embed it in its application window, however we also provide simple and commonly used API for manipulating the window.

/\*\*

\* This structure describes video window info.

\*/

typedef struct pjsua\_vid\_win\_info

{

/\*\*

\* Renderer device ID.

\*/

pjmedia\_vid\_dev\_index rdr\_dev;

/\*\*

\* Native window handle.

\*/

pjmedia\_vid\_dev\_hwnd hwnd;

/\*\*

\* Window show status. The window is hidden if false.

\*/

pj\_bool\_t show;

/\*\*

\* Window position.

\*/

pjmedia\_coord pos;

/\*\*

\* Window size.

\*/

pjmedia\_rect\_size size;

} pjsua\_vid\_win\_info;

/\*\*

\* Enumerates all video windows.

\*

\* @param id Array of window ID to be initialized.

\* @param count On input, specifies max elements in the array.

\* On return, it contains actual number of elements

\* that have been initialized.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_enum\_wins(pjsua\_vid\_win\_id wids[],

unsigned \*count);

/\*\*

\* Get window info.

\*

\* @param wid The video window ID.

\* @param wi The video window info to be initialized.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_win\_get\_info(pjsua\_vid\_win\_id wid,

pjsua\_vid\_win\_info \*wi);

/\*\*

\* Show or hide window.

\*

\* @param wid The video window ID.

\* @param show Set to PJ\_TRUE to show the window, PJ\_FALSE to

\* hide the window.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_win\_set\_show(pjsua\_vid\_win\_id wid,

pj\_bool\_t show);

/\*\*

\* Set video window position.

\*

\* @param wid The video window ID.

\* @param pos The window position.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_win\_set\_pos(pjsua\_vid\_win\_id wid,

const pjmedia\_coord \*pos);

/\*\*

\* Resize window.

\*

\* @param wid The video window ID.

\* @param size The new window size.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_win\_set\_size(pjsua\_vid\_win\_id wid,

const pjmedia\_rect\_size \*size);

### Video Codec API

API for managing video codecs:

/\*\*

\* Enum all supported video codecs in the system.

\*

\* @param id Array of ID to be initialized.

\* @param count On input, specifies max elements in the array.

\* On return, it contains actual number of elements

\* that have been initialized.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_enum\_codecs( pjsua\_codec\_info id[],

unsigned \*count );

/\*\*

\* Change video codec priority.

\*

\* @param codec\_id Codec ID, which is a string that uniquely identify

\* the codec (such as "H263/90000"). Please see pjsua

\* manual or pjmedia codec reference for details.

\* @param priority Codec priority, 0-255, where zero means to disable

\* the codec.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_codec\_set\_priority( const pj\_str\_t \*codec\_id,

pj\_uint8\_t priority );

/\*\*

\* Get video codec parameters.

\*

\* @param codec\_id Codec ID.

\* @param param Structure to receive video codec parameters.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_codec\_get\_param(

const pj\_str\_t \*codec\_id,

pjmedia\_vid\_codec\_param \*param);

/\*\*

\* Set video codec parameters.

\*

\* @param codec\_id Codec ID.

\* @param param Codec parameter to set. Set to NULL to reset

\* codec parameter to library default settings.

\*

\* @return PJ\_SUCCESS on success, or the appropriate error code.

\*/

PJ\_DECL(pj\_status\_t) pjsua\_vid\_codec\_set\_param(

const pj\_str\_t \*codec\_id,

const pjmedia\_vid\_codec\_param \*param);