
 Personal Open source Business Explore Pricing Blog Support

This repository

Search

Sign in

Sign up



Watch83

Star334

Fork142

Code

Issues10

Pull requests10

Projects0

Wiki

Pulse

Graphs

MEGA C++ SDK

3,545 commits

33 branches

1 release

21 contributors


BSD-2-Clause



















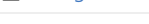

Branch: master


New pull request

Find file

Clone or download

 javiserrano committed on GitHub Merge pull request #421 from meganz/release ... Latest commit 516ee08 5 hours ago

	Add start upload with modification time and temporary source (WP bind...	15 days ago
	included flag for curl in build_android.sh	a month ago
	Definitely remove `MegaApi::addContact()`	5 months ago
	Fixed compilation errors in the Swift example	22 days ago
	New function to upload temporary files with a custom modification time	15 days ago
	Add --with-python3 flag to generate Python 3 language bindings	a year ago
	Scan folder contents after an overwrite	12 days ago
	Merge branch 'develop' of github.com:meganz/sdk into chat-commands	2 months ago
	re(included) glob.c as mega_glob.c	a month ago
	Merge pull request #403 from meganz/master	a month ago
	Add Travis-CI configuration file	2 years ago
	#1 added initial Sphinx and Doxygen documentation and Simplified BSD ...	3 years ago
	moved glob.h to mega_glob.h	a month ago
	Added a missing file to the build script	6 months ago
	Update README.md	4 months ago
	add doxygen support	3 years ago
	* Rename build_static.sh to build_sdk.sh	2 years ago
	moved glob.h to mega_glob.h	a month ago
	reset	2 years ago
	Management of User's avatar	11 months ago



MEGA SDK - Client Access Engine

coverity

passed 1 new defects

MEGA --- *The Privacy Company* --- is a Secure Cloud Storage provider that protects your data thanks to end-to-end encryption. We call it User Controlled Encryption, or UCE, and all our clients automatically manage it.

All files stored on MEGA are encrypted. All data transfers from and to MEGA are encrypted. And while most cloud storage providers can and do claim the same, MEGA is different – unlike the industry norm where the cloud storage provider holds

the decryption key, with MEGA, you control the encryption, you hold the keys, and you decide who you grant or deny access to your files.

This SDK brings you all the power of our client applications and let you create your own or analyze the security of our products. Are you ready to start? Please continue reading.

SDK Contents

In this SDK, you can find our low level SDK, that was already released few months after the MEGA launch, a new intermediate layer to make it easier to use and to bind with other programming languages, and example apps for all our currently supported platforms (Windows, Linux, OSX, Android, iOS and Windows Phone).

In the `examples` folder you can find example apps using:

1. The low level SDK:
 - `megaccli` (a powerful command line tool that allows to use all SDK features)
 - `megasimplesync` (a command line tool that allows to use the synchronization engine)
2. The intermediate layer:
 - An example app for Visual Studio in `examples/win32`
 - An example app for Android (using Java bindings based on SWIG) in `examples/android`
 - An example app for iOS (using Objective-C bindings) in `examples/iOS`
 - An example app for Windows Phone (using Windows Phone bindings) in `examples/wp8`

Building

For platforms with Autotools, the generic way to build and install it is:

```
sh autogen.sh
./configure
make
sudo make install
```

That compilation will include the examples using our low level SDK (`megaccli` and `megasimplesync`) You also have specific build instructions for OSX (`doc/OSX.txt`) and FreeBSD (`doc/FreeBSD.txt`) and a build script to automatically download and build the SDK along with all its dependencies (`contrib/build_sdk.sh`)

For other platforms, or if you want to see how to use the new intermediate layer, the easiest way is to get a smooth start is to build one of the examples in subfolders of the `examples` folder.

All these folders contains a README.md file with information about how to get the project up and running, including the installation of all required dependencies.

Usage

The low level SDK doesn't have inline documentation yet. If you want to use it, please check one of our example apps (`examples/megaccli` , `examples/megasimplesync`).

The new intermediate layer has been documented using Doxygen. The only public header that you need to include to use is `include/megaapi.h`. You can read the documentation in that header file, or download the same documentation in HTML format from this link:

<https://mega.co.nz/#!c5FzhBJL!HUVjsOJTylwkmXPZ0AxT66Wuu4YvZInyHbWGYgvTHt4>

Additional info

Platform Dependencies

Dependencies are different for each platform because the SDK uses generic interfaces to get some features and they have different implementations:

- Network (cURL with OpenSSL/c-ares or WinHTTP)
- Filesystem access (Posix or Win32)
- Graphics management (FreeImage, QT or iOS frameworks)
- Database (SQLite or Berkeley DB)
- Threads/mutexes (Win32, pthread, QT threads, or C++11)

POSIX (Linux/Darwin/BSD/OSX ...)

Install the following development packages, if available, or download and compile their respective sources (package names are for Debian and RedHat derivatives, respectively):

- cURL (`libcurl4-openssl-dev` , `libcurl-devel`), compiled with `--enable-ssl`
- c-ares (`libc-ares-dev` , `libcares-devel` , `c-ares-devel`)
- OpenSSL (`libssl-dev` , `openssl-devel`)
- Crypto++ (`libcrypto++-dev` , `libcryptopp-devel`)
- zlib (`zlib1g-dev` , `zlib-devel`)
- SQLite (`libsqlite3-dev` , `sqlite-devel`) or configure `--without-sqlite`
- FreeImage (`libfreeimage-dev` , `freeimage-devel`) or configure `--without-freeimage`
- pthread

Optional dependency:

- Sodium (`libsodium-dev` , `libsodium-devel`), configure `--with-sodium`

Filesystem event monitoring: The provided filesystem layer implements the Linux `inotify` and the MacOS `fsevents` interfaces.

To build the reference `megacli` example, you may also need to install:

- GNU Readline (`libreadline-dev` , `readline-devel`)

For Android, we provide an additional implementation of the graphics subsystem using Android libraries.

For iOS, we provide an additional implementation of the graphics subsystem using Objective C frameworks.

Windows

To build the client access engine under Windows, you'll need the following:

- A Windows-native C++ development environment (e.g. MinGW or Visual Studio)
- Crypto++
- zlib (until WinHTTP learns how to deal with Content-Encoding: gzip)
- SQLite or configure `--without-sqlite`
- FreeImage or configure `--without-freeimage`
- pthreads (MinGW)

Optional dependency:

- Sodium, configure `--with-sodium`

To build the reference `megacli.exe` example, you will also need to procure development packages (at least headers and `.lib` / `.a` libraries) of:

- GNU Readline/Termcap

Folder syncing

In this version, the sync functionality is limited in scope and functionality:

- There is no locking between clients accessing the same remote folder. Concurrent creation of identically named files and folders can result in server-side dupes.
- Syncing between clients with differing filesystem naming semantics can lead to loss of data, e.g. when syncing a folder containing `ABC.TXT` and `abc.txt` with a Windows client.
- On POSIX platforms, filenames are assumed to be encoded in UTF-8. Invalid byte sequences can lead to undefined behaviour.
- Local filesystem items must not be exposed to the sync subsystem more than once. Any dupes, whether by nesting syncs or through filesystem links, will lead to unexpected results and loss of data.
- No in-place versioning. Deleted remote files can be found in `//bin/SyncDebris` (only when syncing to the logged in account's own cloud drive - there is no SyncDebris facility on syncs to inbound shares), deleted local files in a sync-specific hidden debris folder located in the local sync's root folder.
- No delta writes. Changed files are always overwritten as a whole, which means that it is not a good idea to sync e.g. live database tables.
- No direct peer-to-peer syncing. Even two machines in the same local subnet will still sync via the remote storage infrastructure.
- No support for unidirectional syncing (backup-only, restore-only). Syncing to an inbound share requires it to have full access rights.

`megacli` on Windows

The `megacli` example is currently not handling console Unicode input/output correctly if run in `cmd.exe`.

Filename caveats: Please prefix all paths with `\\?\` to avoid the following issues:

- The `MAX_PATH` (260 character) length limitation, which would make it impossible to access files in deep directory structures
- Prohibited filenames (`con` / `prn` / `aux` / `clock$` / `nul` / `com1` ... `com9` / `lpt1` ... `lpt9`). Such files and folders will still be inaccessible through e.g. Explorer!

Also, disable automatic short name generation to eliminate the risk of clashes with existing short names.

