ownCloud Client Manual

Release 1.4.0

The ownCloud developers

September 04, 2013

CONTENTS

CHAPTER

ONE

INTRODUCTION

This is the documentation for the ownCloud Sync Client, also referred to as the ownCloud Client.

The ownCloud Sync Client is a desktop program you install on your computer. Specify one ore more directories on the local machine to sync your ownCloud server, and always have your latest files wherever you are. Make a change to the files on one computer, it will flow across the others using these desktop sync clients.

ownCloud Client is available for Windows, Mac OS X and various Linux distributions. See below for details on how to obtain the Client.

1.1 Obtaining the Client

The latest version of the ownCloud Client can be obtained at http://owncloud.org/sync-clients/.

ownCloud client for **Windows** is provided as a NSIS-based setup file for machine-wide install. Installing the ownCloud client on **Mac OS** follows the normal app bundle installation pattern:

1. Download the installation file: Click `ownCloud-x.y.z.dmg`, a window with the ownCloud icon opens.
2. In that window, drag the ownCloud application into the `Applications` folder.
3. On the right hand side From `Applications`, choose `ownCloud`.

The ownCloud Client is also provided as in a convenient repository for a wide range of popular **Linux distributions**. If you want to build the sources instead.

Supported distributions are Fedora, openSUSE, Ubuntu and Debian. To support other distributions, a is required, see *Appendix A: Building the Client*

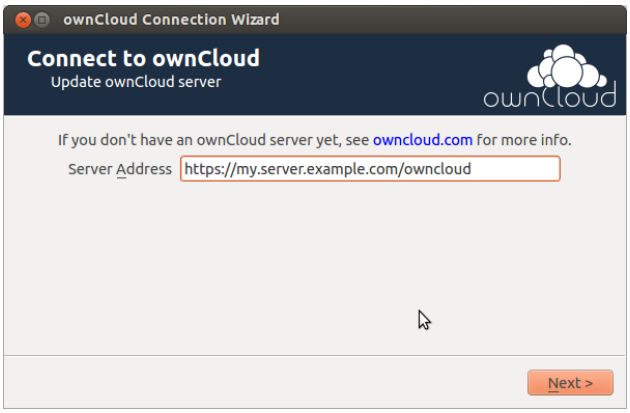
CHAPTER

TWO

SETTING UP AN ACCOUNT

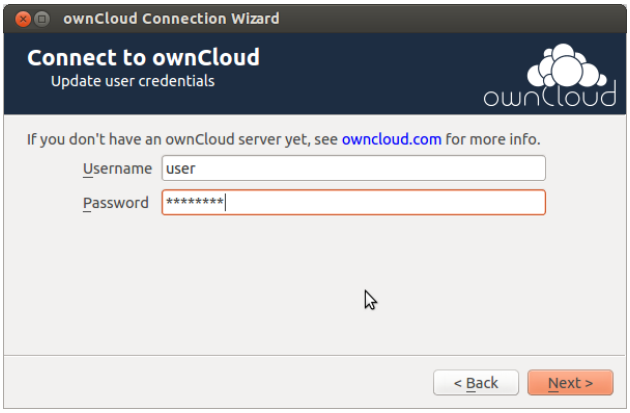
If no account has been configured, ownCloud Client will automatically assist you in connecting to your ownCloud Server after the application has been started.

As a first step, specify the URL to your Server, just like you would when you open your ownCloud instance inside a browser.

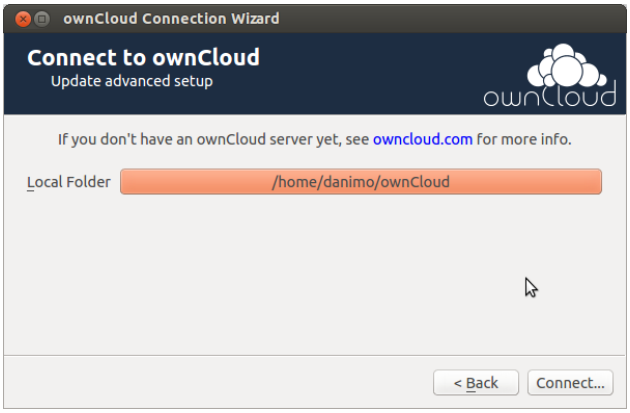


Note: Make sure to use `https://` if the server supports it. Otherwise, your password and all data will be transferred to the server unencrypted. This makes it easy for third parties to intercept your communication, and getting hold of your password!

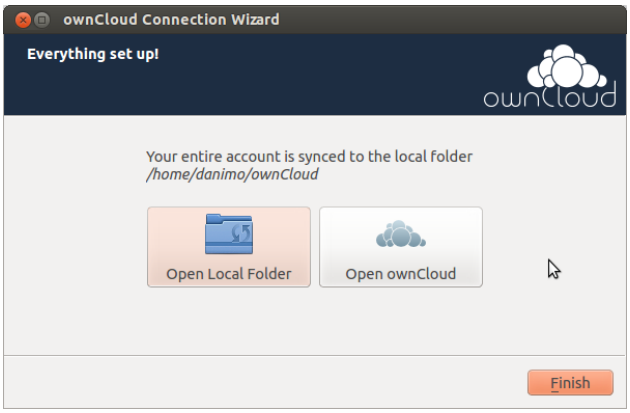
Next, you are prompted for your username and password. Again, use the same credentials that you would use to log on via the web interface.



Finally, choose the folder that ownCloud Client is supposed to sync the contents of your ownCloud account with. By default, this is a folder called *ownCloud*, which will reside in your home directory.



After pressing *Connect*, ownCloud Client will commence with the syncing process. The next screen will give you the opportunity to review your settings:



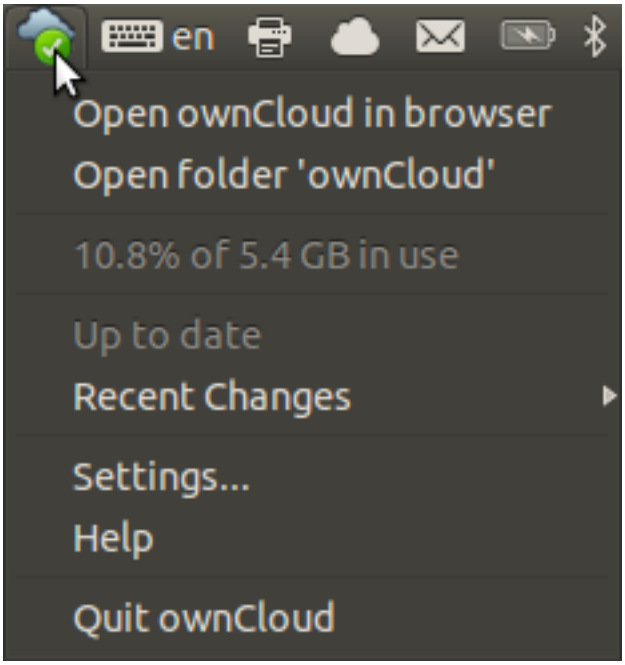
CHAPTER

THREE

VISUAL TOUR

ownCloud Client stays in the background, and is visible as an icon in your system tray (Windows, KDE), status bar (Mac OS X) or notification area (Ubuntu), like so:

If a setup is still required, it will open the setup. Otherwise, the main menu is opened, which provides several options and displays progress information:



Here is an explanation of the individual items in the menu:

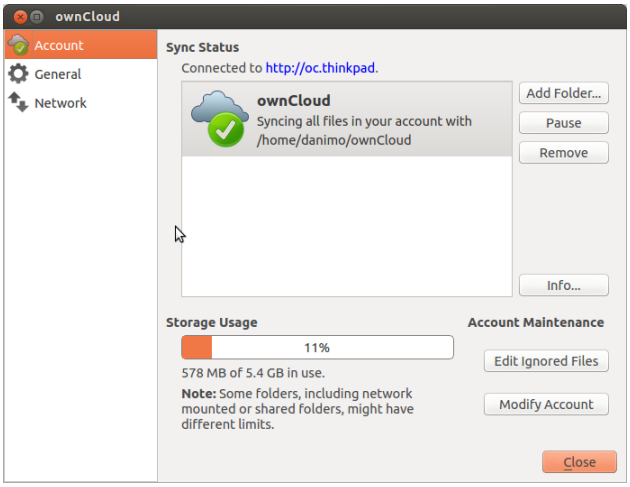
- `Open ownCloud in browser`: Opens the ownCloud web interface
- `Open folder 'ownCloud'`: Opens the local folder. If you have defined multiple sync targets, you should see multiple entries here.
- **Disk space indicator**: Shows how much space is used up on the server.
- Operation indicator: Shows the status of the current sync process, or `Up to date` if server and client are in sync.

- **Recent Changes**: shows the last six files modified by sync operations, and provides access to the Sync Protocol, which lists all changes since the last restart of ownCloud Client.
- `Settings...`: provides access to the settings menu.
- `Help`: Opens a browser to display this help.
- `Quit ownCloud`: Quits ownCloud, ending a currently running sync run.

The settings dialog is split up in three categories: `Account Settings`, `General Settings` and `Network Settings`:

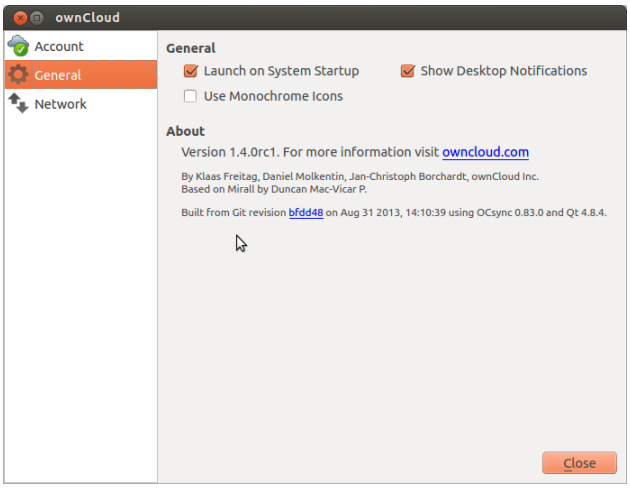
3.1 Account Settings

The `Account Settings` tab provides an executive summary about the synced folders in your account and allows to modify them. It also provides a more detailed report about the storage usage. Finally, it allows to change the files that ownCloud Client should ignore (for details, see the `Ignored Files Editor` section below), and to modify various aspects of the current account settings, such as user name, password and server URL.



3.2 General Settings

The tab provides several useful options:

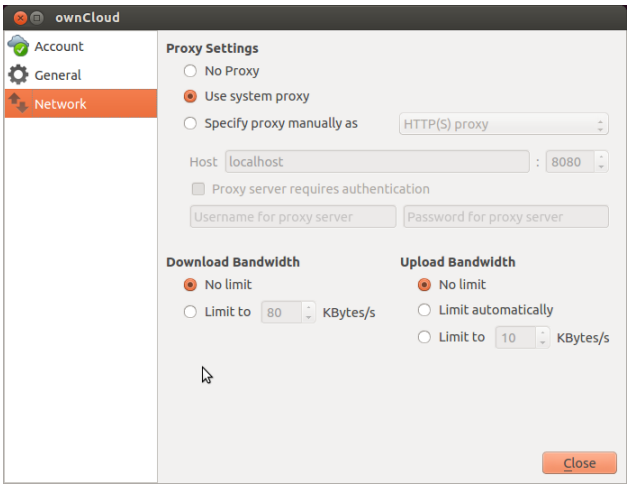


- **Launch on System Startup**: This option is automatically activated once a user has conimaged his account. Unchecking the box will cause ownCloud client to not launch on startup for a particular user.
- **Show Desktop Nofications**: Do not show bubble notifications whenever a set of sync operations has been performed.
- **Use Monochrome Icons**: Use less obstrusive icons. Especially useful on Mac OS.

The acout menu provides information about authors as well as detailed information about the build conditions. Those are particularly valuable when filing a bug report.

3.3 Network Settings

This tab consollidates `Proxy Settings` and `Bandwith Limiting`:



3.3.1 Proxy Settings

- `No Proxy`: Check this if ownCloud Client should circumvent the default proxy conimaged on the system.
- `Use system proxy`: Default, will follow the systems proxy settings. On Linux, this will only pick up the value of the variable `http_proxy`.

- `Specify proxy manually as`: Allows to specify custom proxy settings. If you require to go through a HTTP(S) proxy server such as Squid or Microsoft Forefront TMG, pick `HTTP(S)`. `SOCKSv5` on the other hand is particulary useful in special company LAN setups, or in combination with the OpenSSH dynamic application level forwarding feature (see `ssh -D`).
- `Host`: Enter the host name or IP address of your proxy server, followed by the port number. HTTP proxies usually listen on Ports 8080 (default) or 3128. SOCKS server usually listen on port 1080.
- `Proxy Server requires authentication`: Should be checked if the proxy server does not allow anonymous usage. If you check this option, you must provide username and password in the fields below, or ownless Cloud will no longer be able to connect successfully.

3.3.2 Bandwidth Limiting

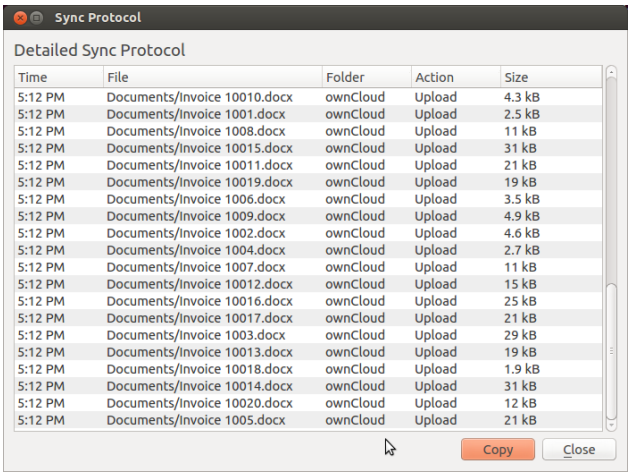
The `Download Bandwidth` (i.e. the bandwidth available for data flowing from the ownCloud Server to the client) can be either `Unlimited` (the default), or limited to a custom value, specified in bytes

The `Upload Bandwith` (i.e. the bandwith available for data flowing from the ownCloud Client to the server) additionally has the option to `Limit automatically`: When this option is checked, the ownCloud Client will surrender available upstream bandwith to other applications. Use this option if you expirience problems with real time communication, such as Skype or other VoIP software, in conjunction with ownCloud Client. This is commonly the case with asymmetric internet connection, such as certain DSL lines with very limited upstream capacity.

ownCloud Client will pick up changes immediately, but ongoing operations will finish using the old settings.

3.4 The Sync Protocol

The `Sync Protocol` window, which can be invoked from either from the main menu (`Recent Changes` -> `Details...`) or the `Account Settings` (`Info` button), will provide you with an in-depth summary of the recent sync activity. It will also show files that have not been synched (ignored files). Those are ignored either because they are listed in the ignored files list (see `Ignored Files Editor` section below), or because they cannot be synced in a cross-platform manner because they contain special characters that cannot be stored on certain file systems.



3.5 The Ignored Files Editor

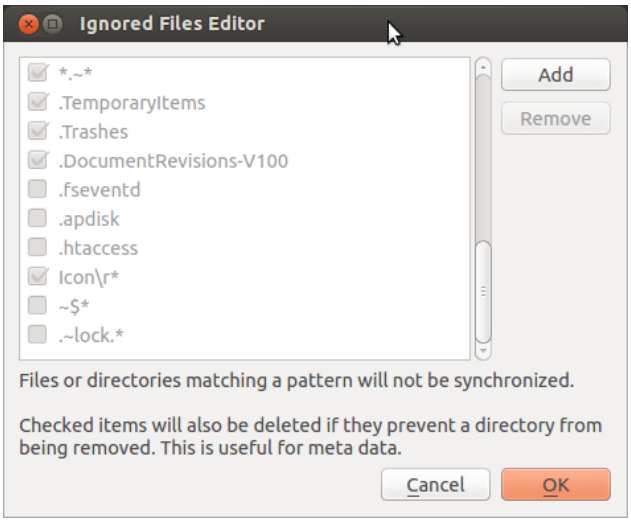
The ignored files editor allows adding patterns for files or directories that should be excluded from the sync process. Next to normal characters, wildcards can be used to match an arbitrary number of characters, designated by an asterisk (*) or a single character, designated by a question mark (?).

Global defaults cannot be directly modified within the editor. Hovering with the mouse will reveal the location of the global exclude definition file.

In addition to this list, ownCloud Client always excludes files with characters that cannot be synched down to other file systems, see *Ignored Files*.

Note: Modifying the global exclude definition file might render the client unusable or cause undesired behavior.

Note: Custom entries are currently not validated for syntactical correctness by the editor, but might fail to load correctly.



3.5.1 Examples:

Pattern	Matches
`~$*`	`~$foo`, `~$example.doc`
`fl?p`	`flip`, `flap`

CHAPTER

FOUR

ADVANCED USAGE

4.1 Options

ownCloud Client supports the following command line switches:

`--logwindow` open a window to show log output.

`--logfile <`*filename*`>` write log output to file <filename>.

`--logdir <`*name*`>` write each sync log output in a new file in directory <name>

`--logexpire <`*hours*`>` removes logs older than <hours> hours. (to be used with –logdir)

`--logflush` flush the log file after every write.

`--confdir <`*dirname*`>` Use the given configuration directory.

4.2 Config File

ownCloud Client reads a configuration file.

On Linux it can be found in: `$HOME/.local/share/data/ownCloud/owncloud.cfg`

On Windows it can be found in: `%LOCALAPPDATA%\ownCloud\owncloud.cfg`

On Mac it can be found in: `$HOME/Library/Application Support/ownCloud`

It contains settings in the ini file format known from Windows.

Note: Changes here should be done carefully as wrong settings can cause disfunctionality.

Note: Changes may be overwritten by using ownCloud's configuration dialog.

These are config settings that may be changed:

`remotePollinterval` (default: `30000`) Poll time for the remote repository in milliseconds

`maxLogLines` (default: `20000`) Maximum count of log lines shown in the log window

CHAPTER

FIVE

APPENDIX A: BUILDING THE CLIENT

This section explains how to build the ownCloud Client from source for all major platforms. You should read this section if you want to development on the desktop client.

5.1 Linux

1. Add the ownCloud repository from OBS.
2. Install the dependencies (as root, or via sudo):

- Debian/Ubuntu: `apt-get update; apt-get build-dep owncloud-client`
- openSUSE: `zypper ref; zypper si -d owncloud-client`
- Fedora/CentOS: `yum install yum-utils; yum-builddep owncloud-client`

3. Follow the generic build instructions.

5.2 Mac OS X

Next to XCode (and the command line tools!), you will need some extra dependencies.

You can install these dependencies via MacPorts or Homebrew. This is only needed on the build machine, since non-standard libs will be deployed in the app bundle.

The tested and preferred way is to use HomeBrew. The ownCloud team has its own repository which contains non-standard recipes. Add it with:

```
brew tap owncloud/owncloud
```

Next, install the missing dependencies:

```
brew install $(brew deps ocsync)
brew install $(brew deps mirall)
```

To build mirall and csync, follow the generic build instructions.

Note: You should not call `make install` at any time, since the product of the mirall build is an app bundle. Call `make package` instead to create an install-ready disk image.

5.3 Windows (cross-compile)

Due to the amount of dependencies that csync entails, building the client for Windows is **currently only supported on openSUSE**, by using the MinGW cross compiler. You can set up openSUSE 12.1 or 12.2 in a virtual machine if you do not have it installed already.

In order to cross-compile, the following repositories need to be added via YaST or `zypper ar` (adjust when using openSUSE 12.2):

```
zypper ar http://download.opensuse.org/repositories/isv:/ownCloud:/devel:/mingw:/win32/openSUSE_12.1/
zypper ar http://download.opensuse.org/repositories/windows:/mingw:/win32/openSUSE_12.1/windows:mingw
zypper ar http://download.opensuse.org/repositories/windows:/mingw/openSUSE_12.1/windows:mingw.repo
```

Next, install the cross-compiler packages and the cross-compiled dependencies:

```
zypper install cmake make mingw32-cross-binutils mingw32-cross-cpp mingw32-cross-gcc \
               mingw32-cross-gcc-c++ mingw32-cross-pkg-config mingw32-filesystem \
               mingw32-headers  mingw32-runtime site-config mingw32-iniparser-devel \
               mingw32-libsqlite-devel mingw32-dlfcn-devel mingw32-libssh2-devel \
               kdewin-png2ico mingw32-libqt4 mingw32-libqt4-devel mingw32-libgcrypt \
               mingw32-libgnutls mingw32-libneon mingw32-libneon-devel mingw32-libbeecrypt \
               mingw32-libopenssl mingw32-openssl mingw32-libpng-devel mingw32-libsqlite \
               mingw32-qtkeychain mingw32-qtkeychain-devel mingw32-iniparser mingw32-dlfcn \
               mingw32-libintl-devel mingw32-libneon-devel mingw32-libopenssl-devel \
               mingw32-libproxy-devel mingw32-libxml2-devel mingw32-zlib-devel
```

For the installer, the NSIS installer package is also required:

```
zypper install mingw32-cross-nsis
```

You will also need to manually download and install the following files with `rpm -ivh <package>` (They will also work with OpenSUSE 12.2):

```
rpm -ihv http://pmbs.links2linux.org/download/mingw:/32/openSUSE_12.1/x86_64/mingw32-cross-nsis-plugi
rpm -ihv http://pmbs.links2linux.org/download/mingw:/32/openSUSE_12.1/x86_64/mingw32-cross-nsis-plugi
```

Now, follow the generic build instructions, but pay attention to the following differences:

1. For building `libocsync`, you need to use `mingw32-cmake` instead of cmake.
2. for building `mirall`, you need to use `cmake` again, but make sure to append the following parameter:
3. Also, you need to specify *absolute pathes* for `CSYNC_LIBRARY_PATH` and `CSYNC_LIBRARY_PATH` when running `cmake` on mirall.

   ```
   -DCMAKE_TOOLCHAIN_FILE=../mirall/admin/win/Toolchain-mingw32-openSUSE.cmake
   ```

Finally, just build by running `make`. `make package` will produce an NSIS-based installer, provided the NSIS mingw32 packages are installed.

5.4 Generic Build Instructions

The ownCloud Client requires Mirall and CSync. Mirall is the GUI frontend, while CSync is responsible for handling the actual synchronization process.

At the moment, ownCloud Client requires a forked version of CSync. Both CMake and Mirall can be downloaded at ownCloud's Client Download Page.

If you want to build the leading edge version of the client, you should use the latest versions of Mirall and CSync via Git, like so:

```
git clone git://git.csync.org/users/freitag/csync.git ocsync
git clone git://github.com/owncloud/mirall.git
```

Next, create build directories:

```
mkdir ocsync-build
mkdir mirall-build
```

This guide assumes that all directories are residing next to each other. Next, make sure to check out the 'dav' branch in the newly checked out *ocsync* directory:

```
cd ocsync
git checkout dav
```

The first package to build is CSync:

```
cd ocsync-build
cmake -DCMAKE_BUILD_TYPE="Debug" ../ocsync
make
```

You probably have to satisfy some dependencies. Make sure to install all the needed development packages. You will need `iniparser`, `sqlite3` as well as `neon` for the ownCloud module. Take special care about `neon`. If that is missing, the cmake run will succeed but silently not build the ownCloud module. `libssh` and `libsmbclient` are optional and not required for the client to work. If you want to install the client, run `make install` as a final step.

Next, we build mirall:

```
cd ../mirall-build
cmake -DCMAKE_BUILD_TYPE="Debug" ../mirall \
      -DCSYNC_BUILD_PATH=/path/to/ocsync-build \
      -DCSYNC_INCLUDE_PATH=/path/to/ocsync/src
```

Note that it is important to use absolute pathes for the include- and library directories. If this succeeds, call `make`. The owncloud binary should appear in the `bin` directory. You can also run `make install` to install the client to `/usr/local/bin`.

To build an installer/app bundle (requires the mingw32-cross-nsis packages on Windows):

```
make package
```

Known cmake parameters:

- WITH_DOC=TRUE: create doc and manpages via running `make`; also adds install statements to be able to install it via `make install`.

CHAPTER

SIX

APPENDIX B: ARCHITECTURE

The ownCloud project provides desktop sync clients to synchronize the contents of local directories on the desktop machines to the ownCloud.

The syncing is done with csync, a bidirectional file synchronizing tool which provides both a command line client as well as a library. A special module for csync was written to synchronize with ownCloud's built-in WebDAV server.

The ownCloud sync client is based on a tool called mirall initially written by Duncan Mac Vicar. Later Klaas Freitag joined the project and enhanced it to work with ownCloud server.

ownCloud Client is written in C++ using the Qt Framework. As a result, the ownCloud Client runs on the three important platforms Linux, Windows and MacOS.

6.1 The Sync Process

First it is important to recall what syncing is: It tries to keep the files on two repositories the same. That means if a file is added to one repository it is going to be copied to the other repository. If a file is changed on one repository, the change is propagated to the other repository. Also, if a file is deleted on one side, it is deleted on the other. As a matter of fact, in ownCloud syncing we do not have a typical client/server system where the server is always master.

This is the major difference to other systems like a file backup where just changes and new files are propagated but files never get deleted.

The ownCloud Client checks both repositories for changes frequently after a certain time span. That is refered to as a sync run. In between the local repository is monitored by a file system monitor system that starts a sync run immediately if something was edited, added or removed.

6.2 Sync by Time versus ETag

Until the release of ownCloud 4.5 and ownCloud Client 1.1, ownCloud employed a single file property to decide which file is newer and hence needs to be synced to the other repository: the files modification time.

The *modification timestamp* is part of the files metadata. It is available on every relevant filesystem and is the natural indicator for a file change. Modification timestamps do not require special action to create and have a general meaning. One design goal of csync is to not require a special server component, that's why it was chosen as the backend component.

To compare the modification times of two files from different systems, it is needed to operate on the same base. Before version 1.1.0, csync requires both sides running on the exact same time, which can be achieved through enterprise standard NTP time synchronisation on all machines.

Since this strategy is rather fragile without NTP, ownCloud 4.5 introduced a unique number, which changes whenever the file changes. Although it is a unique value, it is not a hash of the file, but a randomly chosen number, which it will transmit in the Etag field. Since the file number is guaranteed to change if the file changes, it can now be used to determine if one of the files has changed.

Note: ownCloud Client 1.1 and newer require file ID capabilities on the ownCloud server, hence using them with a server earlier than 4.5.0 is not supported.

Before the 1.3.0 release of the client the sync process might create faux conflict files if time deviates. The original and the conflict files only differed in the timestamp, but not in content. This behaviour was changed towards a binary check if the files are different.

Just like files, directories also hold a unique id, which changes whenever one of the contained files or directories gets modified. Since this is a recursive process, it significantly reduces the effort required for a sync cycle, because the client will only walk directories with a modified unique id.

This table outlines the different sync methods attempted depending on server/client combination:

Server Version	Client Version	Sync Methods
4.0.x or earlier	1.0.5 or earlier	Time Stamp
4.0.x or earlier	1.1 or later	n/a (incompatible)
4.5 or later	1.0.5 or earlier	Time Stamp
4.5 or later	1.1 or later	File ID, Time Stamp

It is highly recommended to upgrade to ownCloud 4.5 or later with ownCloud Client 1.1 or later, since the time stamp-based sync mechanism can lead to data loss in certain edge-cases, especially when multiple clients are involved and one of them is not in sync with NTP time.

6.3 Comparison and Conflict Cases

In a sync run the client first has to detect if one of the two repositories have changed files. On the local repository, the client traverses the file tree and compares the modification time of each file with the value it was before. The previous value is stored in the client's database. If it is not, it means that the file has been added to the local repository. Note that on the local side, the modificaton time a good attribute to detect changes because it does not depend on time shifts and such.

For the remote (ie. ownCloud) repository, the client compares the ETag of each file with it's previous value. Again the previous value is queried from the database. If the ETag is still the same, the file has not changed.

In case a file has changed on both, the local and the remote repository since the last sync run, it can not easily be decided which version of the file is

> the one that should be used. However, changes to any side must not be lost.

That is called a **conflict case**. The client solves it by creating a conflict file of the older of the two files and save the newer one under the original file name. Conflict files are always created on the client and never on the server. The conflict file has the same name as the original file appended with the timestamp of the conflict detection.

6.4 Ignored Files

ownCloud Client will refuse to sync the following files:

- Files matched by one of the pattern in *The Ignored Files Editor*

- Files containing characters that do not work on certain file systems. Currently, these characters are: , :, ?, *, ", >, <, |
- Files starting in `.csync_journal.db` (reserved for journalling)

6.5 The Sync Journal

The client stores the ETag number in a per-directory database, called the journal. It is a hidden file right in the directory to be synced.

If the journal database gets removed, ownCloud Client's CSync backend will rebuild the database by comparing the files and their modification times. Thus it should be made sure that both server and client synchronized with NTP time before restarting the client after a database removal.

Pressing `F5` in the Account Settings Dialog that allows to "reset" the journal. That can be used to recreate the journal database. Use this only if advised to do so by the developer or support staff.

CHAPTER

SEVEN

APPENDIX C: TROUBLESHOOTING

If the client fails to start syncing it basically can have two basic reasons: Either the server setup has a problem or the client has a bug. When reporting bugs, it is crucial to find out what part of the system causes the problem.

7.1 Identifying basic functionality problems

Perform a general ownCloud Server test A very first check is to verify that you can log on to ownClouds web application. Assuming your ownCloud instance is installed at `http://yourserver.com/owncloud`, type `http://yourserver.com/owncloud/` into your browsers address bar.

If you are not prompted to enter your user name and password, or if you see a red warning box on the page, your server setup is not correct or needs fixes. Please verify that your server installation is working correctly.

Ensure the WebDAV API is working If all desktop clients fail to connect to ownCloud, but the access via the web interface works, the problem often is a mis-configuration of the WebDAV API.

The ownCloud client uses the built-in WebDAV access of the server content. Verify that you can log on to ownClouds WebDAV server. Assuming your ownCloud instance is installed at `http://yourserver.com/owncloud`, type `http://yourserver.com/owncloud/remote.php/webdav` into your browsers address bar.

If you are prompted, but the authentication fails even though the credentials your provided are correct, please ensure that your authentication backend is configured properly.

Use a WebDAV command line tool to test A more sophisticated test is to use a WebDAV command line client and log into the ownCloud WebDAV server, such as a little app called cadaver, available on Linux. It can be used to further verify that the WebDAV server is running properly, for example by performing PROPFIND calls:

`propget .` called within cadaver will return some properties of the current directory and thus be a successful WebDAV connect.

7.2 Isolating other issues

If the sync result is unreliable, please ensure that the folder synced with ownCloud is not shared with other syncing apps.

Note: Syncing the same directory with ownCloud and other sync software such as Unison, rsync, Microsoft Windows Offline Folders or cloud services such as DropBox or Microsoft SkyDrive is not supported and should not be attempted. In the worst case, doing so can result in data loss.

If some files do not get take a look at the sync protocol. Some files are automatically automatically being ignored because they are system files, others get ignored because their file name contains characters that cannot be represented on certain file systems. See *_ignored-files-label* for details.

If you are operating your own server and use the local storage backend (the default), make sure that ownCloud has exclusive access to the directory.

Note: The data directory on the server is exclusive to ownCloud and must not be modified manually.

If you are using a different backend, you can try to exclude a bug in the backend by reverting to the local backend.

7.3 Logfiles

Doing effective debugging requires to provide as much as relevant logs as possible. The log output can help you with tracking down problem, and if you report a bug, you're advised to include the output.

7.3.1 Client Logfile

Start the client with `--logwindow`. That opens a window providing a view on the current log. It provides a Save button to let you save the log to a file.

You can also open a log window for an already running session, by simply starting the client again with this parameter. Syntax:

- Windows: `C:\Program Files (x86)\ownCloud\owncloud.exe --logwindow`
- Mac OS X: `/Applications/owncloud.app/Contents/MacOS/owncloud --logwindow`
- Linux: `owncloud --logwindow`

It is also possible to directly log to a directory, which is an useful option in case the problem only happens ocassionally. In that case it is better to create a huge amount of data, as the log window has a limited buffer.

To write logs to disk, start the client with `--logfile <file>`, where `<file` is the file you want to log to, or `--logdir <dir>`, where `<dir>` is an existing directory. In case of `--logdir`, each sync run will create a new file. To limit the amount of data that accumulates over time, there is another useful parameter: `--logexpire <hours>`'. If that is combined with '`--logdir`' the client automatically erases log data in that directory that is older than the given expiry period.

For example, for a long running test where you intend to keep the log data of the last two days, this would be the command line:

```
` owncloud --logdir /tmp/owncloud_logs --logexpire 48 `
```

7.3.2 ownCloud server Logfile

The ownCloud server maintains an ownCloud specific logfile as well. It can and must be enabled through the ownCloud Administration page. There you can adjust the loglevel. It is advisable to set it to a verbose level like `Debug` or `Info`.

The logfile can be viewed either in the web interface or can be found in the filesystem in the ownCloud server data dir.

7.3.3 Webserver Logfiles

Also, please take a look at your webservers error log file to check if there are problems. For Apache on Linux, the error logs usually can be found at `/var/log/apache2`. A file called `error_log` shows errors like PHP code problems. A file called `access_log` usually records all requests handled by the server. Especially the access_log is a very good debugging tool as the log line contains a lot of information of every request and it's result.

More information about the apache logging can be found at `http://httpd.apache.org/docs/current/logs.html`.

CHAPTER

EIGHT

GLOSSARY

mtime, modification time, file modification time File property used to determine whether the servers' or the clients' file is more recent. Standard procedure in oCC 1.0.5 and earlier, used by oCC 1.1 and later only when no sync database exists and files already exist in the client directory.

ownCloud Server The server counter part of ownCloud Client as provided by the ownCloud community.

ownCloud Sync Client, ownCloud Client Name of the official ownCloud syncing client for desktop, which runs on Windows, Mac OS X and Linux. It is based Mirall, and uses the CSync sync engine for synchronization with the ownCloud server.

unique id, ETag ID assigned to every file starting with ownCloud server 4.5 and submitted via the HTTP `Etag`. Used to check if files on client and server have changed.

INDEX