ownCloud Client Manual

Release

The ownCloud developers

September 16, 2014

CHAPTER

ONE

INTRODUCTION

Available for Windows, MAC OS X, and various Linux distributions, the ownCloud Sync client is a desktop program installed on your computer. The client enables you to:

- Specify one or more directories on your computer that you want to synchronize to the ownCloud server.
- Always have the latest files synchronized, wherever they are located.

Changes made to any synchronized file on the computer are automatically made to the files on the ownCloud server using the sync client.

CHAPTER

TWO

INSTALLING THE SYNCHRONIZATION CLIENT

The latest version of the ownCloud Synchronization Client can be obtained from the ownCloud Website. You can download and install the client on Windows, MAC OSX, and various Linux software distrubutions. The following sections describe specific support and installation procedures for the different software platforms:

- *installing-windows*
- *installing-macosx*
- *installing-linux*

CHAPTER

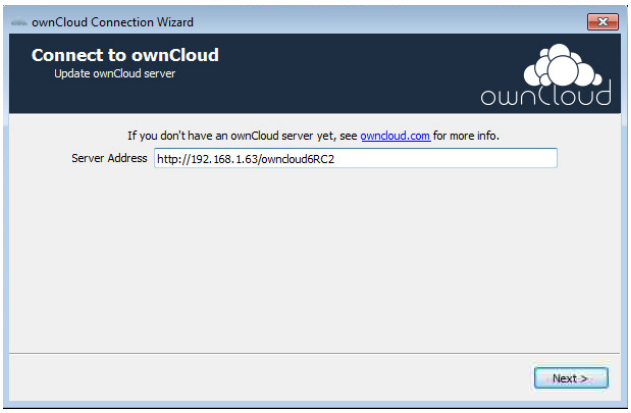
THREE

SETTING UP AN ACCOUNT

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

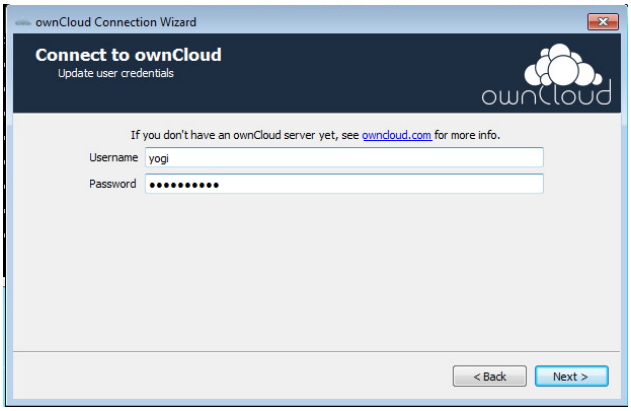
To set up an account:

1. Specify the URL to your Server. This is the same address that is used in the browser.

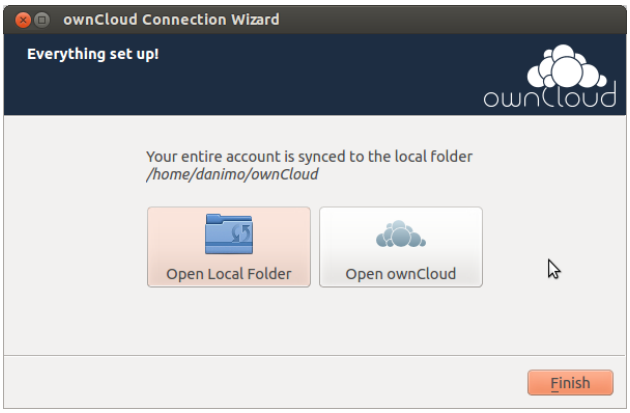


Note: Make sure to use `https://` if the server supports it. Otherwise, your password and all data will be transferred to the server unencrypted. This makes it easy for third parties to intercept your communication, and getting hold of your password!

2. Enter the username and password. These are the same credentials used to log into the web interface.



3. Choose the folder with which you want the ownCloud Client to synchronize the contents of your ownCloud account. By default, this is a folder called *ownCloud*. This folder is created in the home directory.



When selecting a local folder that already contains data, you can choose from two options:

- *Keep local data*: When selected, the files in the local folder on the client are synchronized to the ownCloud server.
- *Start a clean sync*: When selected, all files in the local folder on the client are deleted. These files are not syncrhonized to the ownCloud server.

CHAPTER

FOUR

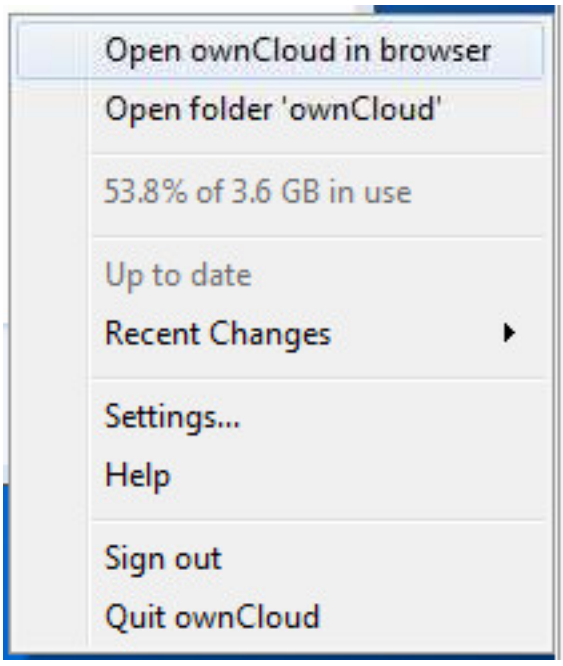
USING THE SYNCHRONIZATION CLIENT

The ownCloud Client remains in the background and is visible as an icon in the system tray (Windows, KDE), status bar (MAC OS X), or notification area (Ubuntu).

ownCloud Desktop Client icon

4.1 Using the Desktop Client Menu

A right click on the icon (left click on Ubuntu and Mac OS X) provides the following menu:



ownCloud Desktop Client menu

The Desktop Client menu provides the following options:

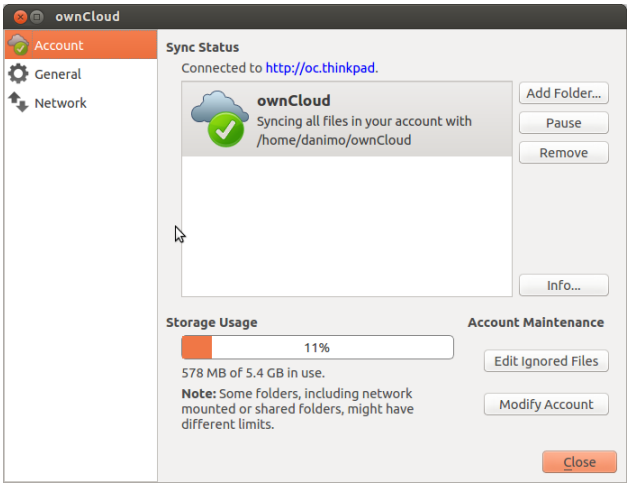
- `Open ownCloud in browser`: Launches the ownCloud WEB interface.
- `Open folder 'ownCloud'`: Opens the ownCloud local folder. If you have defined multiple synchronization targets, the window displays each local folder.

- **Disk space indicator**: Indicates the amount of space currently used on the server.
- Operation indicator: Displays the status of the current synchronization process or indicates `Up to date` if the server and client are in sync.
- **Recent Changes**: Displays the last six files modified by the synchronization operations and provides access to the current synchronization status listing all changes since the last restart of the ownCloud client.
- `Settings...`: Provides access to the settings menu.
- `Help`: Opens a browser to display ownCloud Desktop Client Guide.
- `Sign out`: Disables the client from continued synchronizations.
- `Quit ownCloud`: Quits the ownCloud Client, ending any currently running synchronizations.

4.2 Using the Account Settings Window

The `Account` window provides a summary for general settings associated with the ownCloud account. This window enalbes you to manage any synchronized folders in the account and enables you to modify them.

To access and modify the account settings:



The fields and options in this window include:

- `Connected to <ownCloud instance> as <user>` field: Indicates the ownCloud server to which the client is synchronizing and the user account on that server.
- `Add Folder...` button: Provides the ability to add another folder to the synchronization process (see `Adding a Folder`).
- `Pause/Resume` button: Pauses the current sync (or prevents the client from starting a new sync) or resumes the sync process.
- `Remove` button: Removes the selected folder from the sync process. This button is used when you want to synchronize only a few folders and not the root folder. If only the root folder is available, you must first remove the root from the synchronization and then add individual folders that you want to synchronize as desired.
- `Storage Usage` field: Indicates the storage utilization on the ownCloud server.
- `Edit Ignored Files` button: Launches the Ignored Files Editor.
- `Modify Account` button: Enables you to change the ownCloud server to which you are synchronizing. This option launches the `Setting up an Account` windows (See ??).

4.2.1 Adding a Folder

The `Add a Folder ...` button enables you to add a new folder to the syncrhonization process.

To add a new folder:

1. Click the `Add a Folder ...` button in the Account window.

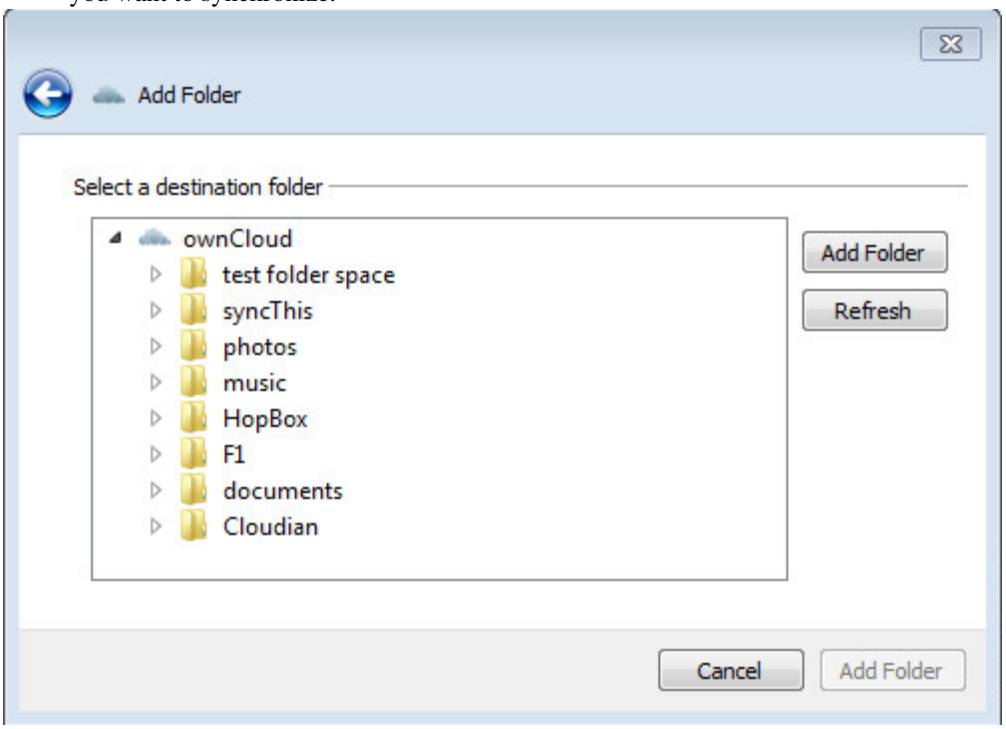
   The `Add Folder...` window opens

2. Specify a *unique* path and alias name to the folder or use the `Choose...` button to locate the new folder on your system to which you want to synchronize.

   ..note:: Nested synchronizations are not supported. In other words, you cannot add a folder that is already contained within another synchronized folder. In addition, you cannot add a higher level (parent) folder that contains a folder to which you are already synchronizing. By default, the ownCloud Set Up Wizard syncrhonizes your entire ownCloud account to the root folder of the ownCloud server. Due to this default setup, you must first remove the top-level folder prior to specifying new synchronizations.

3. Click 'Next' to continue.

   A window opens prompting you to select a remote destination folder on the ownCloud server to which you want to synchronize.



scale 50 %

''Add Folder...'' window (remote destination)

4. Select a folder on the ownCloud server to which you want to synchronize your newly added folder.

   ..note:: A server folder can only be synchronized with a particular client once. If you attempt to sync the root directory, you cannot sync with other folders on the server. Similarly, if you sync with folder `/a`, you cannot create another sync with `/a/b`, since `b` is already being synched.

4.2.2 Editing Ignored Files

The *Ignored Files Editor* provides a list of preconfigured files that are ignored (that is, not synchronized) by the client and server during synchronizations. The Ignored Files Editor enables you to add patterns for files or directories that you want to exclude from the synchronization process. In addition to using standard characters, the Ignored Files Editor enables you to use wild cards (for example, using an asterisk '*' to indicate multiple characters or a question mark '?' to incidate a single character).

For additional information about this editor, see Using the Ignored Files Editor

4.3 Using the Activity Settings Window

The Activity window provides an in-depth account of recent synchronization activity. It shows files that have not been synchronized because they are on the ignored files list or because they cannot be synced in a cross-platform manner due to containing special characters that cannot be stored on certain file systems.

You can open the Activity window in one of the following ways:

- Click 'Activity' in the left frame of the ownCloud Settings window.
- Invoke the window from the ownCloud Desktop Client menu by selecting `Recent Changes` > `Details...`. (See Using the Desktop Client Menu.)

4.4 Using the General Settings Window

The General settings window enables you to set general settings for the ownCloud Desktop Client and provides information about the software version, its creator, and the existance of any updates.

The settings and information contained in this window are as follows:

- `Launch on System Startup` checkbox: Provides the option to check (enable) or uncheck (disable) whether the ownCloud Desktop Client launches upon system startup. By default, this option is enabled (checked)once you have configured your account.
- `Show Desktop Nofications` checkbox: Provides the option to check (enable) or uncheck (disable) bubble notifications alerting you as to when a set of synchronization operations is performed.
- **`Use Monochrome Icons` checkbox: Provides the option to check (enable) or** uncheck (disable) the use of monochrome (visually less obtrusive) icons.

  Note: This option can be useful on MAC OSX platforms.

- `About` field: Provides information about the software authors along with pertinent build conditions.

  Note: Information in this field can be valuable when submitting a support request.

- `Updates` field: Provides information about any available updates for the ownCloud Desktop Client.

4.5 Using the Network Settings Window

The Network settings window enables you to define network proxy settings as well as limit the download and upload bandwidth utilization of file synchronizations.

4.5.1 Specifying Proxy Settings

A proxy server is a server (for example, a computer system or an application) that functions as an intermediary contact for requests from clients that are seeking resources from other servers. For the ownCloud Desktop Client, you can define the following proxy settings:

- `No Proxy` option: Specifies that the ownCloud Client circumvent the default proxy configured on the system.
- `Use system proxy` option: Default setting. Follows the systems proxy settings. On Linux systems, this setting uses the value of the variable `http_proxy`.
- `Specify proxy manually as` option: Enables you to specify the following custom proxy settings: - `HTTP(S)`: Used when you are required to use an HTTP(S) proxy server (for example, Squid or Microsoft Forefront TMG). - `SOCKSv5`: Typically used in special company LAN setups, or in combination with the OpenSSH dynamic application level forwarding feature (see `ssh -D`). - `Host`: Host name or IP address of the proxy server along with the port number. HTTP proxies

  typically listen over Ports 8080 (default) or 3128. SOCKS servers typically listen over port 1080.
- `Proxy Server requires authentication` checkbox: Provides the option to check (enable/require) or uncheck (disable/not require) proxy server authentication. When not checked, the proxy server must be configured to allow anonymous usage. When checked, a proxy server username and password is required.

4.5.2 Bandwidth Limiting

Synchronization of files between a client and server can utilized a lot of bandwidth. Bandwidth limiting can assist in shaping the total download or upload bandwidth (or both) of your client/server connection to a more manageable level. By limiting your bandwidth usage, you can maintain free bandwidth for other applications to use.

The ownCloud Desktop Client enables you to limit (throttle) the bandwidth usage for both file downloads and file uploads. The Download Bandwidth field (for data flowing from the ownCloud server to the client) provides the following options:

- `No limit` option: The default setting for the client; specifies that there are no limit settings on the amount of data downloaded from the server.
- `Limit to <value> KBytes/s` option: Limits (throttles) the bandwidth to a customized value (in KBytes/second).

The Upload Bandwidth field (for data flowing from the ownCloud client to the server) provides the following options:

- `No limit` option: The default setting for the client; specifies that there are no limit settings on the amount of data downloaded from the server.
- `Limit automatically`: When enabled, the ownCloud client surrenders available bandwidth to other applications. Use this option if there are issues with real time communication (for example, the use of IP phone or live streaming) in conjunction with the ownCloud Client.
- `Limit to <value> KBytes/s` option: Limits (throttles) the bandwidth to a customized value (in KBytes/second).

4.6 Using the Ignored Files Editor

You might have some files or directories that you do not want to backup and store on the server. To identify and exclude these files or directories, you can use the *Ignored Files Editor* that is embedded in the ownCloud Desktop Client.

The *Ignored Files Editor* enables you to define customized patterns that the ownCloud Client uses to identify files and directories that you want to exclude from the synchronization process. For your convenience, the editor is pre-populated with a default list of typically ignore patterns. These patterns are contained in a system file (typically `sync-exclude.lst`) located in the ownCloud Client application directory. You cannot modify these pre-populated patterns directly from the editor. However, if necessary, you can hover over any pattern in the list to show the path and filename associated with that pattern, locate the file, and edit the `sync-exclude.lst` file.

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

Each line in the editor contains an ignore pattern string. When creating custom patterns, in addition to being able to use normal characters to define an ignore pattern, you can use wildcards characters for matching values. As an example, you can use an asterisk (`*`) to idenfify an arbitrary number of characters or a question mark (`?`) to identify a single character.

Patterns that end with a slash character (`/`) are applied to only directory components of the path being checked.

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

Each pattern string in the list is preceded by a checkbox. When the check box contains a check mark, in addition to ignoring the file or directory component matched by the pattern, any matched files are also deemed "fleeting metadata" and removed by the client.

In addition to excluding files and directories that use patterns defined in this list:

- The ownCloud Client always excludes files containing characters that cannot be synchronized to other file systems.
- As of ownCloud Desktop Client version 1.5.0, files are removed that cause individual errors three times during a synchronization. However, the client provides the option of retrying a synchronization three additional times on files that produce errors.

For more detailed information see *Ignored Files*.

CHAPTER

FIVE

ADVANCED USAGE

5.1 Options

When invoking the client from the command line, the following options are supported:

-h, --help Displays all the options below or, when used on Windows, opens a window displaying all options.

--logwindow Opens a window displaying log output.

--logfile <*filename*> Write log output to the file specified. To write to stdout, specify - as the filename.

--logdir <*name*> Writes each synchronization log output in a new file in the specified directory.

--logexpire <*hours*> Removes logs older than the value specified (in hours). This command is used with `--logdir`.

--logflush Clears (flushes) the log file after each write action.

--confdir <*dirname*> Uses the specified configuration directory.

5.2 Config File

The ownCloud Client reads a configuration file. You can locate this configuration files as follows:

- **On Linux distributions:** `$HOME/.local/share/data/ownCloud/owncloud.cfg`
- **In Microsoft Windows systems:** `%LOCALAPPDATA%\ownCloud\owncloud.cfg`
- **In MAC OS X systems:** `$HOME/Library/Application Support/ownCloud`

The configuration file contains settings using the Microsoft Windows .ini file format. You can overwrite changes using the ownCloud configuration dialog.

Note: Use caution when making changes to the ownCloud Client configuration file. Incorrect settings can produce unintended results.

You can change the following configuration settings:

- `remotePollInterval` (default: `30000`) – Specifies the poll time for the remote repository in milliseconds.
- `maxLogLines` (default: `20000`) – Specifies the maximum number of log lines displayed in the log window.

5.3 ownCloud Commandline Client

The ownCloud Client packages contain a command line client that can be used to synchronize ownCloud files to client machines. The command line client is called `owncloudcmd`.

owncloudcmd performs a single *sync run* and then exits the synchronization process. In this manner, owncloudcmd processes the differences between client and server directories and propagates the files to bring both repositories to the same state. Contrary to the GUI-based client, owncloudcmd does not repeat synchronizations on its own. It also does not monitor for file system changes.

To invoke the owncloudcmd, you must provide the local and the remote repository urls using the following command:

```
owncloudcmd [OPTIONS...] sourcedir owncloudurl
```

where `sourcedir` is the local directory and `owncloudurl` is the server URL.

Note: Prior to the 1.6 version of owncloudcmd, the tool only accepted `owncloud://` or `ownclouds://` in place of `http://` and `https://` as a scheme. See `Examples` for details.

Other comand line switches supported by owncloudcmd include the following:

- **`--silent`** Supresses verbose log output.
- **`--confdir`** ***PATH*** Fetches or stores configuration in the specified configuration directory.
- **`--httpproxy http://[user@pass:]<server>:<port>`** Uses the specified `server` as the HTTP proxy.

5.3.1 Credential Handling

By default, owncloudcmd reads the client configuration and uses the credentials of the GUI syncrhonization client. If no client is configured, or if you choose to use a different user to synchronize, you can specify the user password setting with the usual URL pattern. For example:

```
https://user:secret@192.168.178.2/remote.php/webdav
```

5.3.2 Example

To synchronize the ownCloud directory `Music` to the local directory `media/music`, through a proxy listening on port ``8080`, and on a gateway machine using IP address `192.168.178.1`, the command line would be:

```
$ owncloudcmd --httpproxy http://192.168.178.1:8080 \
              $HOME/media/music \
              https://server/owncloud/remote.php/webdav/Music
```

Using the legacy scheme, the command line would be:

```
$ owncloudcmd --httpproxy http://192.168.178.1:8080 \
              $HOME/media/music \
              ownclouds://server/owncloud/remote.php/webdav/Music
```

CHAPTER

SIX

THE AUTOMATIC UPDATER

To ensure that you are always using the latest version of the ownCloud client, an auto-update mechanism has been added in Version 1.5.1. The Automatic Updater ensures that you automatically profit from the latest features and bugfixes.

Note: The Automatic Updater functions differently, depending on the operating system.

6.1 Basic Workflow

The following sections describe how to use the Automatic Updater on different operating systems:

6.1.1 Windows

The ownCloud client checks for updates and downloads them when available. You can view the update status under `Settings -> General -> Updates` in the ownCloud client.

If an update is available, and has been successfully downloaded, the ownCloud client starts a silent update prior to its next launch and then restarts itself. Should the silent update fail, the client offers a manual download.

Note: Administrative privileges are required to perform the update.

6.1.2 Mac OS X

If a new update is available, the ownCloud client initializes a pop-up dialog to alert you of the update and requesting that you update to the latest version. Due to their use of the Sparkle frameworks, this is the default process for Mac OS X applications.

6.1.3 Linux

Linux distributions provide their own update tool, so ownCloud clients that use the Linux operating system do not perform any updates on their own. Linux operating systems do, however, check for the latest version of the ownCloud client and passively notify the user (`Settings -> General -> Updates`) when an update is available.

6.2 Preventing Automatic Updates

In controlled environments, such as companies or universities, you might not want to enable the auto-update mechanism, as it interferes with controlled deployment tools and policies. To address this case, it is possible to disable the auto-updater entirely. The following sections describe how to disable the auto-update mechanism for different operating systems.

6.2.1 Preventing Automatic Updates in Windows Environents

You can prevent automatic updates from occuring in Windows environments using one of two methods. The first method allows users to override the automatic update check mechanism whereas the second method prevents any manual overrides.

To prevent automatic updates, but allow manual overrides:

1. Migrate to the following directory:

   `HKEY_LOCAL_MACHINE\Software\ownCloud\ownCloud`

2. Add the key `skipUpdateCheck` (of type DWORD).
3. Specify a value of `1` to the machine.

To manually override this key, use the same value in `HKEY_CURRENT_USER`.

To prevent automatic updates and disallow manual overrides:

1. Migrate to the following directory:

   `HKEY_LOCAL_MACHINE\Software\Policies\ownCloud\ownCloud`

2. Add the key `skipUpdateCheck` (of type DWORD).
3. Specify a value of `1` to the machine.

6.2.2 Preventing Automatic Updates in Mac OS X Environments

You can disable the automatic update mechanism in MAC OS X operating systems using the system-wide `.plist` file. To access this file:

1. Using the Windows explorer, migrate to the following location:

   `/Library/Preferences/`

2. Locate and open the following file:

   `com.owncloud.desktopclient.plist`

3. Add a new root level item of type `bool`.
4. Name the item `skipUpdateCheck`.
5. Set the item to `true`.

Alternatively, you can copy the file `owncloud.app/Contents/Resources/deny_autoupdate_com.owncloud.deskto`
to `/Library/Preferences/com.owncloud.desktopclient.plist`.

6.2.3 Preventing Automatic Updates in Linux Environments

Because Linux does not provide automatic updating functionality, there is no need to remove the automatic-update check. However, if you want to disable this check:

1. Locate and open the following file:

   ```
   /etc/ownCloud/ownCloud.conf
   ```

2. Add the following content to the file:

   ```
   [General]
   skipUpdateCheck=true
   ```

CHAPTER

SEVEN

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 develop for the desktop client.

Note: Building instruction are subject to change as development proceeds. Please check the version for which you want to built.

The instructions contained in this topic were updated to work with version 1.5 of the ownCloud Client.

7.1 Linux

1. Add the ownCloud repository from OBS.
2. Install the dependencies (as root, or using `sudo`) using the following commands for your specific Linux distribution:

- 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.

7.2 Mac OS X

In additon to needing XCode (along with the command line tools), developing in the MAC OS X environment requires extra dependencies. You can install these dependencies through MacPorts or Homebrew. These dependencies are required only on the build machine, because non-standard libs are deployed in the app bundle.

The tested and preferred way to develop in this environment is through the use of HomeBrew. The ownCloud team has its own repository containing non-standard recipes.

To set up your build enviroment for development using HomeBrew:

1. Add the ownCloud repository using the following command:

   brew tap owncloud/owncloud

2. Install any missing dependencies:

   brew install $(brew deps mirall)

To build mirall, follow the generic build instructions.

Note: Because the product from the mirall build is an app bundle, do not call `make install` at any time. Instead, call `make package` to create an install-ready disk image.

7.3 Windows (Cross-Compile)

Due to the large number of dependencies, building the client for Windows is **currently only supported on openSUSE**, by using the MinGW cross compiler. You can set up openSUSE 12.1, 12.2, or 13.1 in a virtual machine if you do not have it installed already.

To cross-compile:

1. Add the following repositories using YaST or `zypper ar` (adjust when using openSUSE 12.2 or 13.1):
   - `zypper ar http://download.opensuse.org/repositories/windows:/mingw:/win32/openSUSE_13.`
   - `zypper ar http://download.opensuse.org/repositories/windows:/mingw/openSUSE_13.1/windo`
2. 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-libqt4-sql mingw32-libqt4-sql-sqlite mingw32-sqlite mingw32-libsqlite-devel mingw32-dlfcn-devel mingw32-libssh2-devel kdewin-png2ico mingw32-libqt4 mingw32-libqt4-devel mingw32-libgcrypt mingw32-libgnutls mingw32-libneon-openssl mingw32-libneon-devel mingw32-libbeecrypt mingw32-libopenssl mingw32-openssl mingw32-libpng-devel mingw32-libsqlite mingw32-qtkeychain mingw32-qtkeychain-devel mingw32-dlfcn mingw32-libintl-devel mingw32-libneon-devel mingw32-libopenssl-devel mingw32-libproxy-devel mingw32-libxml2-devel mingw32-zlib-devel‘‘
3. For the installer, install the NSIS installer package:

   `zypper install mingw32-cross-nsis`
4. Install the following plugin:

   ``mingw32-cross-nsis-plugin-processes mingw32-cross-nsis-plugin-uac``

   Note: This plugin is typically required. However, due to a current bug in `mingw`, the plugins do not currently build properly from source.
5. Manually download and install the following files using `rpm -ivh <package>`:

   ..note:: These files operate using openSUSE 12.2 and newer.
   - `rpm -ihv http://download.tomahawk-player.org/packman/mingw:32/openSUSE_12.1/x86_64/`
   - `rpm -ihv http://download.tomahawk-player.org/packman/mingw:32/openSUSE_12.1/x86_64/`
6. Follow the generic build instructions

   Note: When building for Windows platforms, you must specify a special toolchain file that enables cmake to locate the platform-specific tools. To add this parameter to the call to cmake, enter `DCMAKE_TOOLCHAIN_FILE=../mirall/admin/win/Toolchain-mingw32-openSUSE.cmake`.
7. Build by running `make`.

..note:: Using `make package` produces an NSIS-based installer, provided the NSIS mingw32 packages are installed.

7.4 Generic Build Instructions

Compared to previous versions, building Mirall has become easier. Unlike earlier versions, CSync, which is the sync engine library of Mirall, is now part of the Mirall source repository and not a separate module.

You can download Mirall from the ownCloud Client Download Page.

To build the most up to date version of the client:

1. Clone the latest versions of Mirall from Git as follows:

   ```
   git clone git://github.com/owncloud/mirall.git
   ```

2. Create build directories:

   ```
   mkdir mirall-build
   ```

3. Build mirall:

   ```
   cd ../mirall-build cmake -DCMAKE_BUILD_TYPE="Debug" ../mirall
   ```

   ..note:: You must use absolute pathes for the `include` and `library` directories.

4. Call `make`.

   The owncloud binary appear in the `bin` directory.

5. (Optional) Call `make install` to install the client to the `/usr/local/bin` directory.

6. (Optional) Call `make package` to build an installer/app bundle

   ..note:: This step requires the `mingw32-cross-nsis` packages be installed on Windows.

The following are known cmake parameters:

- **`QTKEYCHAIN_LIBRARY=/path/to/qtkeychain.dylib -DQTKEYCHAIN_INCLUDE_DIR=/path/to/qtkeych`**
  Used for stored credentials. When compiling with Qt5, the library is called `qt5keychain.dylib`. You need to compile QtKeychain with the same Qt version.
- `WITH_DOC=TRUE`: Creates doc and manpages through running `make`; also
- adds install statements, providing the ability to install using ‘‘make
- install‘‘.
- `CMAKE_PREFIX_PATH=/path/to/Qt5.2.0/5.2.0/yourarch/lib/cmake/`: Builds using Qt5.
- `BUILD_WITH_QT4=ON`: Builds using Qt4 (even if Qt5 is found).

CHAPTER

EIGHT

APPENDIX B: HISTORY AND ARCHITECTURE

ownCloud provides desktop sync clients to synchronize the contents of local directories from computers, tablets, and handheld devices to the ownCloud server.

Synchronization is accomplished using csync, a bidirectional file synchronizing tool that provides both a command line client as well as a library. A special module for csync was written to synchronize with the ownCloud 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 function with the ownCloud server.

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

8.1 The Synchronization Process

The process of synchronization keeps files in two separate repositories the same. When syncrhonized:

- If a file is added to one repository it is copied to the other synchronized repository.
- When a file is changed in one repository, the change is propagated to any syncrhonized other repositories- If a file is deleted in one repository, it is deleted in any other.

It is important to note that the ownCloud synchronization process does not use a typical client/server system where the server is always master. This is a major difference between the ownCloud syncrhonizatin process and other systems like a file backup, where only changes to files or folders and the addition of new files are propagated, but these files and folders are never deleted unless explicitly deleted in the backup.

During synchronization, the ownCloud Client checks both repositories for changes frequently. This process is referred to as a *sync run*. In between sync runs, the local repository is monitored by a file system monitoring process that starts a sync run immediately if something was edited, added, or removed.

8.2 Synchronization by Time versus ETag

Until the release of ownCloud 4.5 and ownCloud Client 1.1, the ownCloud synchronization process employed a single file property – the file modificatin time – to decide which file was newer and needed to be synchronized to the other repository.

The *modification timestamp* is part of the files metadata. It is available on every relevant filesystem and is the typical 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. This design goal is why csync was chosen as the backend component.

To compare the modification times of two files from different systems, csync must operate on the same base. Before ownCloud Client version 1.1.0, csync required both device repositories to run on the exact same time. This requirement was achieved through the use of enterprise standard NTP time synchronisation on all machines.

Because this timing strategy is rather fragile without the use of NTP, ownCloud 4.5 introduced a unique number (for each file?) that changes whenever the file changes. Although this number is a unique value, it is not a hash of the file. Instead, it is a randomly chosen number, that is transmitted in the Etag field. Because the file number changes if the file changes, its use is guaranteed to determine if one of the files has changed and, thereby, launching a synchronization process.

Note: ownCloud Client release 1.1 and later requires file ID capabilities on the ownCloud server. Servers that run with release earlier than 4.5.0 do not support using the file ID functionality.

Before the 1.3.0 release of the Desktop Client, the synchronization process might create faux conflict files if time deviates. Original and changed files conflict only in their timestamp, but not in their content. This behaviour was changed to employ a binary check if files differ.

Like files, directories also hold a unique ID that changes whenever one of the contained files or directories is modified. Because this is a recursive process, it significantly reduces the effort required for a synchronization cycle, because the client only analyzes directories with a modified ID.

The following table outlines the different synchronization methods used, 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

We strongly recommend using ownCloud Server release 4.5 or later when using ownCloud Client 1.1 or later. Using incompatible time stamp-based synchronization mechanism can lead to data loss in rare cases, especially when multiple clients are involved and one utilizes a non-synchronized NTP time.

8.3 Comparison and Conflict Cases

As mentioned above, during a *sync run* the client must first 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 an expected value stored in its database. If the value is not the same, the client determines that the file has been modified in the local repository.

Note: On the local side, the modificaton time a good attribute to use for detecting changes, because

---

the value does not depend on time shifts and such.

For the remote (that is, ownCloud server) repository, the client compares the ETag of each file with its expected value. Again, the expected ETag value is queried from the client database. If the ETag is the same, the file has not changed and no synchronization occurs.

In the event 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 be lost. Instead, a *conflict case* is created. The client resolves this conflic by creating a conflict file of the older of the two files and saving the newer file under the original file name. Conflict files are always created on the client and never on the server. The conflict file uses the same name as the original file, but is appended with the timestamp of the conflict detection.

8.4 Ignored Files

The ownCloud Client supports the ability to exclude or ignore certain files from the synchronization process. Some system wide file patterns that are used to exclude or ignore files are included with the client by default and the ownCloud Client provides the ability to add custom patterns.

By default, the ownCloud Client ignores the following files:

- Files matched by one of the patterns defined in *ignoredFilesEditor-label*.
- Files containing characters that do not work on certain file systems (*, :, ?, *, ", >, <,* |).
- Files starting in `.csync_journal.db*`, as these files are reserved for journalling.

If a pattern selected using a checkbox in the *ignoredFilesEditor-label* (or if a line in the exclude file starts with the character *]* directly followed by the file pattern), files matching the pattern are considered *fleeting meta data*. These files are ingored and *removed* by the client if found in the synchronized folder. This is suitable for meta files created by some applications that have no sustainable meaning.

If a pattern ends with the backslash (*/*) character, only directories are matched. The pattern is only applied for directory components of filenames selected using the checkbox.

To match filenames against the exclude patterns, the unix standard C library function fnmatch is used. This procesx checks the filename against the specified pattern using standard shell wildcard pattern matching. For more information, please refer to *The opengroup website <http://pubs.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html#tag_02_13_01>*.

The path that is checked is the relative path under the sync root directory.

Pattern and File Match Examples:

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

8.5 The Sync Journal

The client stores the ETag number in a per-directory database, called the *journal*. This database is a hidden file contained in the directory to be synchronized.

If the journal database is removed, the ownCloud Client CSync backend rebuilds the database by comparing the files and their modification times. This process ensures that both server and client are synchronized using the appropriate NTP time before restarting the client following a database removal.

Pressing `F5` while in the Account Settings Dialog enables you to "reset" the journal. This function can be used to recreate the journal database.

Note: We recommend that you use this function only when advised to do so by ownCloud support staff.

CHAPTER

NINE

APPENDIX C: TROUBLESHOOTING

The following two general issues can result in failed synchronization:

- The server setup is incorrect.
- The client contains a bug.

When reporting bugs, it is helpful if you first determine what part of the system is causing the issue.

9.1 Identifying Basic Functionality Problems

Performing a general ownCloud Server test The first step in troubleshooting synchronization issues is to verify that you can log on to the ownCloud web application. To verify connectivity to the ownCloud server:

- Open a browser window and enter the server address to your own server in the location/address bar.

For example, if your ownCloud instance is installed at URL address `http://yourserver.com/owncloud`, enter `http://yourserver.com/owncloud/` into your browsers location/address bar.

If you are not prompted for your username and password, or if a red warning box appears on the page, your server setup requires modification. Please verify that your server installation is working correctly.

Ensure the WebDAV API is working If all desktop clients fail to connect to the ownCloud Server, but access using the web interface functions properly, the problem is often a misconfiguration 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. To verify connectivity with the ownCloud WebDAV server:

- Open a browser window and enter the address to the ownCloud WebDAV server.

For example, if your ownCloud instance is installed at `http://yourserver.com/owncloud`, your WebDAV server address is `http://yourserver.com/owncloud/remote.php/webdav`.

If you are prompted for your username and password but, after providing the correct credentials, authentication fails, please ensure that your authentication backend is configured properly.

Use a WebDAV command line tool to test A more sophisticated test method for troubleshooting synchronization issues is to use a WebDAV command line client and log into the ownCloud WebDAV server. One such command line client – called cadaver – is available for Linux distributions. You can

use this application to further verify that the WebDAV server is running properly using PROPFIND calls.

As an example, after installing the cadaver app, you can issue the `propget` command to obtain various properties pertaining to the current directory and also verify WebDAV server connection.

9.2 Isolating other issues

Other issues can affect synchronization of your ownCloud files:

- If you find that the results of the synchronizations are unreliable, please ensure that the folder to which you are synchronizing is not shared with other synchronization applications.

  Note: Synchronizing the same directory with ownCloud and other

  synchronization software such as Unison, rsync, Microsoft Windows Offline Folders, or other cloud services such as DropBox or Microsoft SkyDrive is not supported and should not be attempted. In the worst case, it is possible that synchronizing folders or files using ownCloud and other synchronization software or services can result in data loss.

- If you find that only specific files are not synrchronized, the synchronization protocol might be having an effect. Some files are automatically ignored because they are system files, other files might be ignored because their filename contains characters that are not supported on certain file systems. For more information about ignored files, see *_ignored-files-label*.

- 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 file backend on the server, you can try to exclude a bug in the backend by reverting to the built-in backend.

9.3 Log Files

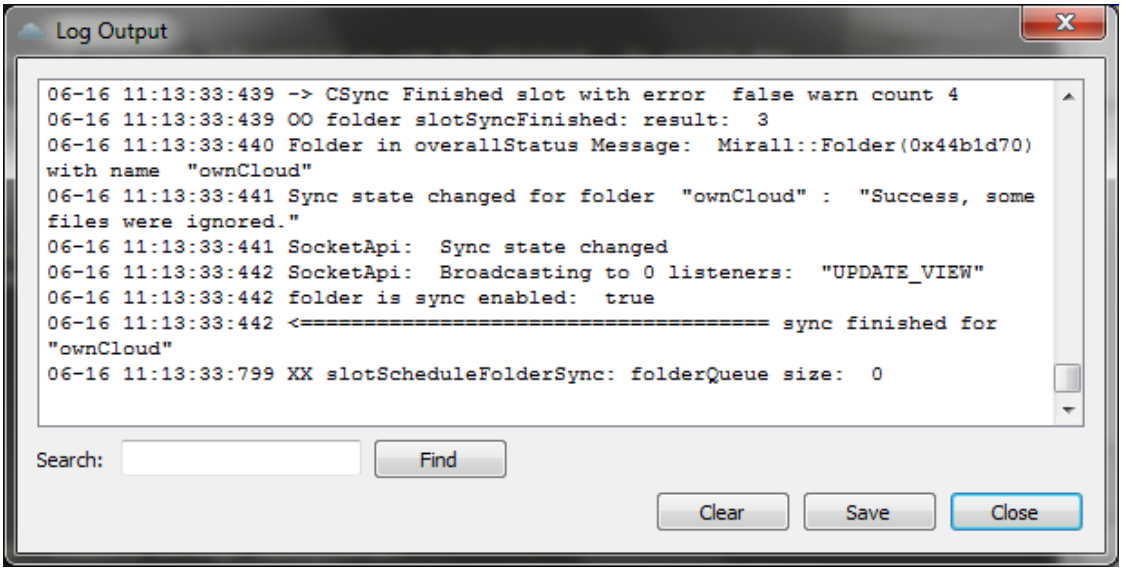
Effectively debugging software requires as much relative information as can be obtained. To assist the ownCloud support personnel, please try to provide as many relevant logs as possible. Log output can help with tracking down problems and, if you report a bug, log output can help to resolve an issue more quickly.

9.3.1 Obtaining the Client Log File

To obtain the client log file:

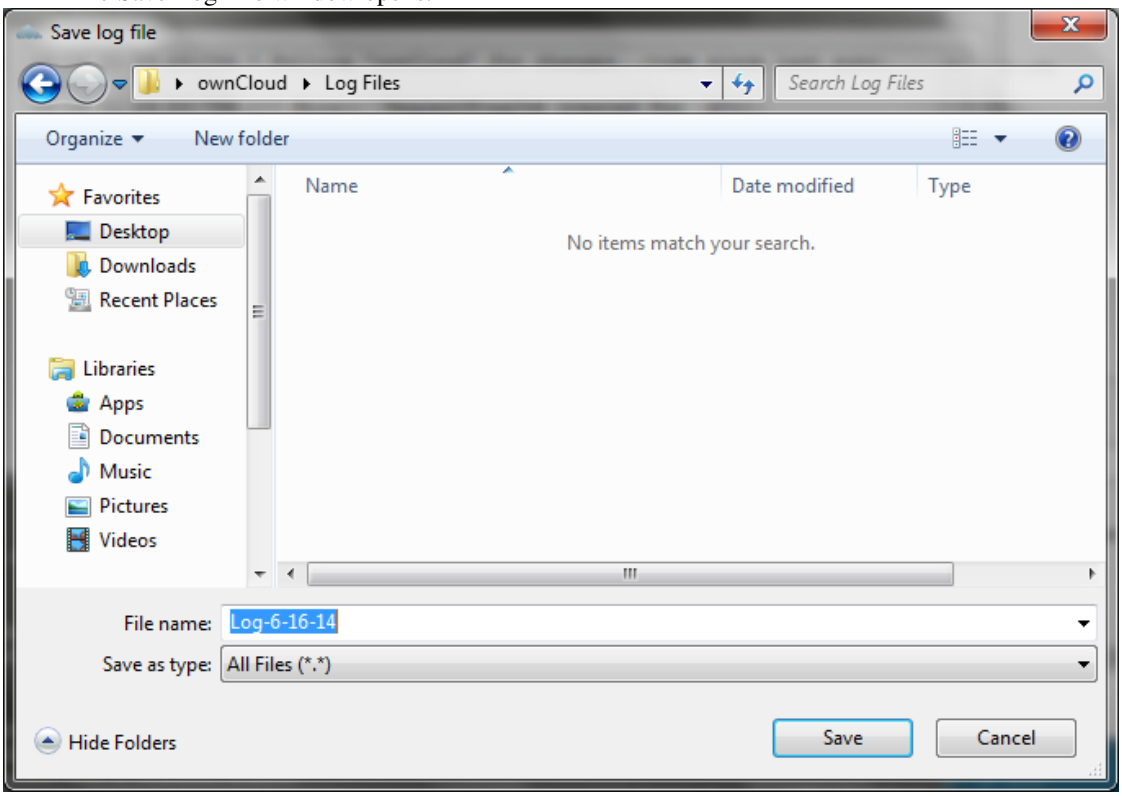
1. Open the ownCloud Desktop Client.
2. Press F12 on your keyboard.

   The Log Output window opens.



3. Click the 'Save' button.

   The Save Log File window opens.



4. Migrate to a location on your system where you want to save your log file.
5. Name the log file and click the 'Save' button.

   The log files is saved in the location specifed.

Alternatively, you can launch the ownCloud Log Output window using the `--logwindow` command. After issuing this command, the Log Output window opens to show the current log. You can then follow the same procedures mentioned above to save the log to a file.

Note: You can also open a log window for an already running session, by restarting the client using the following command:

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

9.3.2 Saving Files Directly

The ownCloud client enables you to save log files directly to a predefined file or directory. This is a useful option for troubleshooting sporadic issues as it enables you to log large amounts of data and bypasses the limited buffer settings associated with the log window.

To save log files to a file or a directory:

1. To save to a file, start the client using the `--logfile <file>` command, where `<file>` is the filename to which you want to save the file.
2. To save to a directory, start the client using the `--logdir <dir>` command, where `<dir>` is an existing directory.

When using the `--logdir` command, each sync run creates a new file. To limit the amount of data that accumulates over time, you can specify the `--logexpire <hours>` command. When combined with the `--logdir` command, the client automatically erases saved log data in the directory that is older than the specified number of hours.

As an example, to define a test where you keep log data for two days, you can issue the following command:

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

9.3.3 ownCloud server Log File

The ownCloud server also maintains an ownCloud specific log file. This log file must be enabled through the ownCloud Administration page. On that page, you can adjust the log level. We recommend that when setting the log file level that you set it to a verbose level like `Debug` or `Info`.

You can view the server log file using the web interface or you can open it directly from the file system in the ownCloud server data directory.

Todo

Need more information on this. How is the log file accessed? Need to explore procedural steps in access and in saving this file ... similar to how the log file is managed for the client. Perhaps it is detailed in the Admin Guide and a link should be provided from here. I will look into that when I begin heavily editing the Admin Guide.

9.3.4 Webserver Log Files

It can be helpful to view your webservers error log file to isolate any ownCloud-related problems. For Apache on Linux, the error logs are typically located in the `/var/log/apache2` directory. Some helpful files include the following:

- `error_log` – Maintains errors associated with PHP code.

- `access_log` – Typically records all requests handled by the server; very useful as a debugging tool because the log line contains information specific to each request and its result.

You can find more information about Apache logging at `http://httpd.apache.org/docs/current/logs.html`.

9.4 Core Dumps

On MAC OS X and Linux systems, and in the unlikely event the client software crashes, the client is able to write a core dump file. Obtaining a core dump file can assist ownCloud Customer Support tremendously in the debugging process.

To enable the writing of core dump files, you must define the `OWNCLOUD_CORE_DUMP` environment variable on the system.

For example:

```
` OWNCLOUD_CORE_DUMP=1 owncloud `
```

This command starts the client with core dumping enabled and saves the files in the current working directory.

Note: Core dump files can be fairly large. Before enabling core dumps on your system, ensure that you have enough disk space to accommodate these files. Also, due to their size, we strongly recommend that you properly compress any core dump files prior to sending them to ownCloud Customer Support.

CHAPTER

TEN

FAQ

Issue:

Some files are continuously uploaded to the server, even when they are not modified.

Resolution:

It is possible that another program is changing the modification date of the file.

If the file is uses the `.eml` extention, Windows automatically and continually changes all files, unless you remove ‘‘HKEY_LOCAL_MACHINESOFTWAREMicrosoftWindowsCurrentVersionPropertySystemPropertyHandlers‘ from the windows registry.

See http://petersteier.wordpress.com/2011/10/22/windows-indexer-changes-modification-dates-of-eml-files/ for more information.

CHAPTER

ELEVEN

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.