ownCloud Developer Manual

The ownCloud Team

Version 10.0, April 24, 2020

Table of Contents

ownCloud Developer Documentation 1
General 1
Help and Communication 34
Core Development 34
Introduction 174
Mobile Development 338
Bugtracker 372
Have You Found a Mistake In The Documentation? 378

ownCloud Developer Documentation

If you want to contribute please read the Contributor agreement

app/index	core/index	Documentation
Develop apps for ownCloud and publish on the ownCloud Marketplace	Develop on the ownCloud internals	Create and enhance documentation
testing/index Help us to test ownCloud by joining the testing team	*bugtracker/index* Report, triage or fix bugs to improve quality	*Translation* Translate ownCloud into your language
commun/index Help on IRC, the mailinglist and forum	*mobile_development/ios_library/index* Integration with iOS	*mobile_development/android_library/index* Integrating with Android

General

In this section you will find a range of general information on developing with ownCloud, such as performance, security, debugging, and backporting.

Community Code of Conduct

Preamble:

In the ownCloud community, participants from all over the world come together to create Free Software for a free internet. This is made possible by the support, hard work and enthusiasm of thousands of people, including those who create and use ownCloud software.

This document offers some guidance to ensure ownCloud participants can cooperate effectively in a positive and inspiring atmosphere, and to explain how together we can strengthen and support each other.

This Code of Conduct is shared by all contributors and users who engage with the ownCloud team and its community services.

Overview

This Code of Conduct presents a summary of the shared values and common sense thinking in our community. The basic social ingredients that hold our project together include:

- Be considerate
- Be respectful
- Be collaborative
- Be pragmatic
- Support others in the community

- Get support from others in the community

Our community is made up of several groups of individuals and organizations which can roughly be divided into two groups:

- Contributors, or those who add value to the project through improving ownCloud software and its services
- Users, or those who add value to the project through their support as consumers of ownCloud software

This Code of Conduct reflects the agreed standards of behavior for members of the ownCloud community, in any forum, mailing list, wiki, web site, IRC channel, public meeting or private correspondence within the context of the ownCloud team and its services.

The community acts according to the standards written down in this Code of Conduct and will defend these standards for the benefit of the community. Leaders of any group, such as moderators of mailing lists, IRC channels, forums, etc., will exercise the right to suspend access to any person who persistently breaks our shared Code of Conduct.

Be collaborative

The Free Software Movement depends on collaboration: it helps limit duplication of effort while improving the quality of the software produced. In order to avoid misunderstanding, try to be clear and concise when requesting help or giving it. Remember it is easy to misunderstand emails (especially when they are not written in your mother tongue). Ask for clarifications if unsure how something is meant; remember the first rule – assume in the first instance that people mean well.

As a contributor, you should aim to collaborate with other community members, as well as with other communities that are interested in or depend on the work you do. Your work should be transparent and be fed back into the community when available, not just when ownCloud releases. If you wish to work on something new in existing projects, keep those projects informed of your ideas and progress.

It may not always be possible to reach consensus on the implementation of an idea, so don’t feel obliged to achieve this before you begin. However, always ensure that you keep the outside world informed of your work, and publish it in a way that allows outsiders to test, discuss and contribute to your efforts.

Contributors on every project come and go. When you leave or disengage from the project, in whole or in part, you should do so with pride about what you have achieved and by acting responsibly towards others who come after you to continue the project.

As a user, your feedback is important, as is its form. Poorly thought out comments can cause pain and the demotivation of other community members, but considerate discussion of problems can bring positive results. An encouraging word works wonders.

Be considerate

Your actions and work will affect and be used by other people and you in turn will depend on the work and actions of others. Any decision you take will affect other community members, and we expect you to take those consequences into account when making decisions.

As a contributor, ensure that you give full credit for the work of others and bear in mind how your changes affect others. It is also expected that you try to follow the development schedule and guidelines.

Be pragmatic

ownCloud is a pragmatic community. We value tangible results over having the last word in a discussion. We defend our core values like freedom and respectful collaboration, but we don't let arguments about minor issues get in the way of achieving more important results.

We are open to suggestions and welcome solutions regardless of their origin. When in doubt support a solution which helps getting things done over one which has theoretical merits, but isn't being worked on. Use the tools and methods which help getting the job done. Let decisions be taken by those who do the work.

As a user, remember that contributors work hard on their part of ownCloud and take great pride in it. If you are frustrated your problems are more likely to be resolved if you can give accurate and well-mannered information to all concerned.

Be respectful

In order for the ownCloud community to stay healthy its members must feel comfortable and accepted. Treating one another with respect is absolutely necessary for this. In a disagreement, in the first instance assume that people mean well.

We do not tolerate personal attacks, racism, sexism or any other form of discrimination. Disagreement is inevitable, from time to time, but respect for the views of others will go a long way to winning respect for your own view. Respecting other people, their work, their contributions and assuming well-meaning motivation will make community members feel comfortable and safe and will result in motivation and productivity.

We expect members of our community to be respectful when dealing with other contributors, users and communities. Remember that ownCloud is an international project and that you may be unaware of important aspects of other cultures.

Get support from others in the community

Disagreements, both political and technical, happen all the time. Our community is no exception to the rule. The goal is not to avoid disagreements or differing views but to resolve them constructively. You should turn to the community to seek advice and to resolve disagreements and where possible consult the team most directly involved.

Think deeply before turning a disagreement into a public dispute. If necessary request mediation, trying to resolve differences in a less highly-emotional medium. If you do feel that you or your work is being attacked, take your time to breathe through before writing heated replies. Consider a 24 hour moratorium if emotional language is being used – a cooling off period is sometimes all that is needed. If you really want to go a different way, then we encourage you to publish your ideas and your work, so that it can be tried and tested.

The authors of this document would like to thank the ownCloud community and those who have worked to create such a dynamic environment to share in and who offered their thoughts and wisdom in the authoring of this document. We would also like to thank other vibrant communities that have helped shape this document with their own examples, especially KDE.

Support others in the community

Our community is made strong by mutual respect, collaboration and pragmatic,

responsible behavior. Sometimes there are situations where this has to be defended and other community members need help.

If you witness others being attacked, think first about how you can offer them personal support. If you feel that the situation is beyond your ability to help individually, go privately to the victim and ask if some form of official intervention is needed. Similarly you should support anyone who appears to be in danger of burning out, either through work-related stress or personal problems.

When problems do arise, consider respectfully reminding those involved of our shared Code of Conduct as a first action. Leaders are defined by their actions, and can help set a good example by working to resolve issues in the spirit of this Code of Conduct before they escalate.

Coding Style & General Guidelines

Coding

- Maximum line-length of 80 characters
- Use tabs to indent
- A tab is 4 spaces wide
- Opening braces of blocks are on the same line as the definition
- Quotes: ’ for everything, " for HTML attributes (<p class="my_class">)
- End of Lines : Unix style (LF / \n) only
- No global variables or functions
- Unit tests
- HTML should be HTML5 compliant
- Check these database performance tips
- When you git pull, always git pull --rebase to avoid generating extra commits like: *merged master into master*

CSS

Take a look at the Writing Tactical CSS & HTML video on YouTube.

Don’t bind your CSS too much to your HTML structure and try to avoid IDs. Also try to make your CSS reusable by grouping common attributes into classes.

DO:

```
.list {
   list-style-type: none;
}

.list > .list_item {
   display: inline-block;
}

.important_list_item {
   color: red;
}
```

DON'T:

```
#content .myHeader ul {
   list-style-type: none;
}

#content .myHeader ul li.list_item {
   color: red;
   display: inline-block;
}
```

General

- Ideally, discuss your plans on the mailing list to see if others want to work with you on it
- We use Github, please get an account there and clone the repositories you want to work on
- Fixes go directly to master, nevertheless they need to be tested thoroughly.
- New features are always developed in a branch and only merged to master once they are fully done.
- Software should work. We only put features into master when they are complete. It's better to not have a feature instead of having one that works poorly.
- It is best to start working based on an issue - create one if there is none. You describe what you want to do, ask feedback on the direction you take it and take it from there.
- When you are finished, use the merge request function on Github to create a pull request. The other developers will look at it and give you feedback. You can signify that your PR is ready for review by adding the label 5 - ready for review to it. You can also post your merge request to the mailing list to let people know. See the code review page for more information <../bugtracker/codereviews>
- It is essential to keep changes small and separate. The bigger a PR grows, the harder it is to complete a quick and efficient review. Given that, split larger changes up into smaller changes, where you can. For example, if you need a minor improvement, get it in first rather than adding it as part of a much larger piece of work.
- Decisions are made by consensus. We strive for making the best technical decisions and as nobody can know everything, we collaborate. That means a first negative comment might not be the final word, neither is positive feedback an immediate GO. ownCloud is built out of modular pieces (apps) and maintainers have a strong influence. In case of disagreement we consult other seasoned contributors.

Labels

We assign labels to issues and pull requests to make it easier to find them as well as to signal what needs to be done with them. Some of these are assigned by the developers, others by QA, bug triggers, project lead or maintainers and so on. It is not desired that users/reporters of bugs assign labels themselves, unless they are developers/contributors to ownCloud.

The most important labels and their meaning:

Label	Meaning
#bug	This issue is a bug
#enhancement	This issue is a feature request/idea for improvement of ownCloud
#design	This needs help from the design team or is a design-related issue/pull request
#sharing	This issue or PR is related to sharing
#technical debt	This issue or PR is about technical debt
#sev1-critical #sev2-high #sev3-medium `#sev4-low `	Signify how important the bug is.
#p1-urgent #p2-high #p3-medium #p4-low	Signify the priority of the bug.
#Junior Job	These are issues which are relatively easy to solve and ideal for people who want to learn how to code in ownCloud
#triage	This issue *has to be* triaged
#needs info	This issue needs further information from the reporter, see triaged old tag is #clarification request, please don't use that one anymore.
#discussion	This issue needs to be discussed
#security	This is a security related issue
#windows server	This is related to windows server
#research	This item requires some research before it can continue
#packaging	This is related to packaging
#theming	Refers to theming issues or improvements
#l10n	Refers to translation issues or improvements
#release note	Relevant for the release notes
#privacy	Refers to issues that might lead to privacy concerns
#won't fix	This problem won't be fixed (can be for a wide variety of reasons.)

Tag Groups

Group	Tags	Description
App tags	#app:files #app:user_ldap #app:files_versions and so on.	These tags indicate the app that is impacted by the issue or which the PR is related to
Settings tags	#settings:personal #settings:apps #settings:admin and so on.	These tags indicate the settings area that is impacted by the issue or which the PR is related to

Group	Tags	Description
db tags	#db:mysql #db:sqlite #db:postgresql and so on.	These tags indicate the database that is impacted by the issue or which the PR is related to
Browser tags	#browser:ie #browser:safari and so on.	These tags indicate the browser that is impacted by the issue or which the PR is related to
Component tags	#comp:filesystem #comp:javascript and so on.	These tags indicate the components of ownCloud impacted by the issue or which the PR is related to
Development tool tags	#dev:unit_testing #dev:public_API and so on.	These tags indicate development-specific tools like those for testing and public developer-facing API's impacted by the issue or which the PR is related

Labels showing the state of the issue or PR (numbered 1-6)

Label	Description
#1 - To develop	Ready to start development on this
#2 - Developing	Development in progress
#3 - To Review	Ready for review
#4 - To Release	Reviewed PR that awaits unfreeze of a branch to get merged

Severity Level Labels

To better understand which severity level to apply, if any, here is a description of each of the four severity labels.

Label	Description
#sev1-critical	The operation is in production and is mission critical to the business. The product is inoperable and the situation is resulting in a total disruption of work. There is no workaround available.
#sev2-high	Operations are severely restricted. Important features are unavailable, although work can continue in a limited fashion. A workaround is available.
#sev3-medium	The product does not work as designed resulting in a minor loss of usage. A workaround is available.
#sev4-low	There is no loss of service. This may be a request for documentation, general information, product enhancement request, etc.

Don't See The Label You Need?

If you want a label not in the list above, please first discuss on the mailing list.

JavaScript

In general take a look at JSLint without the whitespace rules.

- Use a js/main.js or js/app.js where your program is started
- Complete every statement with a **;**
- Use **var** to limit variable to local scope
- To keep your code local, wrap everything in a self executing function. To access global objects or export things to the global namespace, pass all global objects to the self executing function.
- Use JavaScript strict mode
- Use a global namespace object where you bind publicly used functions and objects to

DO:

```
// set up namespace for sharing across multiple files
var MyApp = MyApp || {};

(function(window, $, exports, undefined) {
   'use strict';

   // if this function or object should be global, attach it to the namespace
   exports.myGlobalFunction = function(params) {
      return params;
   };

})(window, jQuery, MyApp);
```

DONT (Seriously):

```
// This does not only make everything global but you're programming
// JavaScript like C functions with namespaces
MyApp = {
   myFunction:function(params) {
      return params;
   },
   ...
};
```

Objects & Inheritance

Try to use OOP in your JavaScript to make your code reusable and flexible.

This is how you'd do inheritance in JavaScript:

```
// create parent object and bind methods to it
var ParentObject = function(name) {
   this.name = name;
};

ParentObject.prototype.sayHello = function() {
   console.log(this.name);
}


// create childobject, call parents constructor and inherit methods
var ChildObject = function(name, age) {
   ParentObject.call(this, name);
   this.age = age;
};

ChildObject.prototype = Object.create(ParentObject.prototype);

// overwrite parent method
ChildObject.prototype.sayHello = function() {
   // call parent method if you want to
   ParentObject.prototype.sayHello.call(this);
   console.log('childobject');
};

var child = new ChildObject('toni', 23);

// prints:
// toni
// childobject
child.sayHello();
```

Objects, Functions & Variables

Use Pascal case for Objects, Camel case for functions and variables.

```
var MyObject = function() {
    this.attr = "hi";
};

var myFunction = function() {
    return true;
};

var myVariable = 'blue';

var objectLiteral = {
    value1: 'somevalue'
};
```

Operators

Use **===** and **!==** instead of **==** and **!=**.

Here's why:

```
` == '0'          // false
0 == `            // true
0 == '0'          // true

false == 'false'    // false
false == '0'        // true

false == undefined  // false
false == null       // false
null == undefined   // true

' \t\r\n ' == 0     // true
```

Control Structures

- Always use \{ } for one line ifs
- Split long ifs into multiple lines
- Always use break in switch statements and prevent a default block with warnings if it shouldn't be accessed

DO:

```
// single line if
if (myVar === 'hi') {
    myVar = 'ho';
} else {
    myVar = 'bye';
}

// long ifs
if (   something === 'something'
    || condition2
    && condition3
) {
  // your code
}

// for loop
for (var i = 0; i < 4; i++) {
    // your code
}

// switch
switch (value) {

    case 'hi':
        // yourcode
        break;

    default:
        console.warn('Entered undefined default block in switch');
        break;
}
```

PHP

The ownCloud coding style guide is based on PEAR Coding Standards. To check your PHP codestyle use PHP Code Sniffer >= 3.0 with the phpcs.xml config file from the core branch.

To check one file use: phpcs --standard=./phpcs.xml yourCode.php

To check all files in a folder (recursive) use: phpcs --standard=./phpcs.xml your/code/folder/

A git pre-commit hook is available here. Download and save the file in the .git/hooks folder of your owncloud project and change the PHPCS_STANDARD constant to the path of the phpcs.xml file.

Start & closing

Always use:

```
<?php
```

at the start of your php code. The final closing:

```
?>
```

should not be used at the end of the file due to the possible issue of sending white spaces.

Comments

All API methods need to be marked with PHPDoc markup. An example would be:

```
<?php

/**
 * Description what method does
 * @param Controller $controller the controller that will be transformed
 * @param API $api an instance of the API class
 * @throws APIException if the api is broken
 * @since 4.5
 * @return string a name of a user
 */
public function myMethod(Controller $controller, API $api) {
  // ...
}
```

Objects, Functions, Arrays & Variables

Use Pascal case for Objects, Camel case for functions and variables. If you set a default function/method parameter, do not use spaces. Do not prepend private class members with underscores.

```
class MyClass {

}

function myFunction($default=null) {

}

$myVariable = 'blue';

$someArray = array(
   'foo'  => 'bar',
   'spam' => 'ham',
);

?>
```

Operators

Use **===** and **!==** instead of **==** and **!=**.

Here's why:

```
<?php

var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true
var_dump("10" == "1e1"); // 10 == 10 -> true
var_dump(100 == "1e2"); // 100 == 100 -> true

?>
```

Control Structures

- Always use \{ } for one line ifs
- Split long ifs into multiple lines
- Always use break in switch statements and prevent a default block with warnings if it shouldn't be accessed

```php
<?php

// single line if
if ($myVar === 'hi') {
    $myVar = 'ho';
} else {
    $myVar = 'bye';
}

// long ifs
if (    $something === 'something'
    || $condition2
    && $condition3
) {
  // your code
}

// for loop
for ($i = 0; $i < 4; $i++) {
    // your code
}

switch ($condition) {
    case 1:
        // action1
        break;

    case 2:
        // action2;
        break;

    default:
        // defaultaction;
        break;
}

?>
```

Unit tests

Unit tests must always extend the \Test\TestCase class, which takes care of cleaning up the installation after the test.

If a test is run with multiple different values, a data provider must be used. The name of the data provider method must not start with test and must end with Data.

```php
<?php
namespace Test;
class Dummy extends \Test\TestCase {
   public function dummyData() {
      return array(
         array(1, true),
         array(2, false),
      );
   }

   /**
    * @dataProvider dummyData
    */
   public function testDummy($input, $expected) {
      $this->assertEquals($expected, \Dummy::method($input));
   }
}
```

User Interface

- Software should not get in the way of what the user needs to do. It should do as much as possible automatically, instead of offering configuration options for the user to chose from.
- Software should be easy to use. Show only the most important elements. Secondary elements should only appear as a result of a hovering the mouse over an element, or via choosing advanced functionality.
- User data is sacred. Provide undo instead of asking for confirmation - which might be dismissed
- The state of the application should be clear. If something loads, provide feedback.
- Do not adapt broken concepts (for example design of desktop apps) just for the sake of consistency. We aim to provide a better interface, so let's find out how to do that!
- Regularly reset your installation to see what the first-run experience looks like — then improve it!
- Ideally do usability testing to know how people use the software.
- For further UX principles, read Alex Faaborg from Mozilla.

Debugging

Debugging HTML and templates

By default ownCloud caches HTML generated by templates. This may prevent changes to app templates, for example, from being applied on page refresh. To disable caching, see Debug mode.

Debugging Javascript

By default all JavaScript files in ownCloud are minified (compressed) into a single file without whitespace. To prevent this, see Debug mode.

Debug mode

When debug mode is enabled in ownCloud, a variety of debugging features are enabled - see debugging documentation. Set debug to true in /config/config.php to enable it:

Debugging variables

You should use exceptions if you need to debug variable values manually, and not alternatives like trigger_error() (which may not be logged), e.g.,:

```
<?php throw new \Exception( "\$user = $user" ); // should be logged in ownCloud
?>
```

not:

```
<?php trigger_error( "\$user = $user" ); // may not be logged anywhere ?>
```

To disable custom error handling in ownCloud (and have PHP and your Web server handle errors instead), see Debug mode.

Identifying errors

ownCloud uses custom error PHP handling that prevents errors being printed to Web server log files or command line output. Instead, errors are generally stored in ownCloud's own log file, located at: /data/owncloud.log.

Using alternative app directories

It may be useful to have multiple app directories for testing purposes, so you can conveniently switch between different versions of applications. See the configuration file documentation for details.

Using a PHP debugger (XDebug)

Using a debugger connected to PHP allows you to step through code line by line, view variables at each line and even change values while the code is running. The de-facto standard debugger for PHP is XDebug, available as an installable package in many distributions. It just provides the PHP side however, so you will need a frontend to actually control XDebug. When installed, it needs to be enabled in php.ini, along with some parameters to enable connections to the debugging interface:

XDebug will now (when activated) try to connect to localhost on port 9000, and will communicate over the standard protocol DBGP. This protocol is supported by many debugging interfaces, such as the following popular ones:

- vdebug - Multi-language DBGP debugger client for Vim
- SublimeTextXdebug - XDebug client for Sublime Text
- PhpStorm - in-built DBGP debugger

For further reading, see the XDebug documentation: http://xdebug.org/docs/remote

Once you are familiar with how your debugging client works, you can start debugging with XDebug. To test ownCloud through the web interface or other HTTP requests, set the XDEBUG_SESSION_START cookie or POST parameter. Alternatively, there are browser extensions to make this easy:

- The Easiest XDebug (Firefox): https://addons.mozilla.org/en-US/firefox/addon/the-easiest-xdebug/
- XDebug Helper (Chrome): https://chrome.google.com/extensions/detail/eadndfjplgieldjbigjakmdgkmoaaaoc

For debugging scripts on the command line, like `occ` or unit tests, set the `XDEBUG_CONFIG` environment variable.

Performance Considerations

This document introduces some common considerations and tips on improving performance of ownCloud. Speed of ownCloud is important - nobody likes to wait and often, what is *just slow* for a small amount of data will become *unusable* with a large amount of data. Please keep these tips in mind when developing for ownCloud and consider reviewing your app to make it faster.

Tips welcome: More tips and ideas on performance are very welcome!

Database performance

The database plays an important role in ownCloud performance. The general rule is: database queries are very bad and should be avoided if possible. The reasons for that are:

- Roundtrips: Bigger ownCloud installations have the database not installed on the application server but on a remote dedicated database server. The problem is that database queries then go over the network. These roundtrips can add up significantly if you have a lot of queries.
- Speed. A lot of people think that databases are fast. This is not always true if you compare it with handling data internally in PHP or in the filesystem or even using key/value based storages. So every developer should always double check if the database is really the best place for the data.
- Scalability. If you have a big ownCloud cluster setup you usually have several ownCloud/Web servers in parallel and a central database and a central storage. This means that everything that happens on the ownCloud/PHP side can parallelize and can be scaled. Stuff that is happening in the database and in the storage is critical because it only exists once and can’t be scaled so easily.

We can reduce the load on the database by:

1. Making sure that every query uses an index.
2. Reducing the overall number of queries.
3. If you are familiar with cache invalidation you can try caching query results in PHP.

There a several ways to monitor which queries are actually executed on the database.

With MySQL it is very easy with just a bit of configuration:

1. Slow query log.

If you put this into your my.cnf file, every query that takes longer than one second is logged to a logfile:

```
log_slow_queries = 1
log_slow_queries = /var/log/mysql/mysql-slow.log
long_query_time=1
```

If a query takes more than a second we have a serious problem of course. You can watch it with tail -f /var/log/mysql/mysql-slow.log while using ownCloud.

1. log all queries.

If you reduce the long_query_time to zero then every statement is logged. This is super helpful to see what is going on. Just do a tail -f on the logfile and click around in the interface or access the WebDAV interface:

```
log_slow_queries = 1
log_slow_queries = /var/log/mysql/mysql-slow.log
long_query_time=0
```

1. log queries without an index.

If you increase the long_query_time to 100 and add log-queries-not-using-indexes, all the queries that are not using an index are logged. Every query should always use an index. So ideally there should be no output:

```
log-queries-not-using-indexes
log_slow_queries = 1
log_slow_queries = /var/log/mysql/mysql-slow.log
long_query_time=100
```

Measuring performance

If you do bigger changes in the architecture or the database structure you should always double check the positive or negative performance impact. There are a few nice small scripts that can be used for this.

The recommendation is to automatically do 10000 PROPFINDs or file uploads, measure the time and compare the time before and after the change.

Getting help

If you need help with performance or other issues please ask on our mailing list or on our IRC channel **#owncloud-dev** on **irc.freenode.net**.

Security Guidelines

Introduction

These security guidelines are for both core and application developers. They:

- highlight some of the most common security problems and how to prevent them.
- give you some best practices and tips about security when developing with ownCloud.

Please use them to assess how secure your application is.

Program defensively: for instance always check for CSRF or escape strings, even if you do not need it. Doing so prevents future problems where you might miss a change that leads to a security hole.

All application Framework security features depend on the call of the controller through OCA\\AppFramework\\App::main. If the controller method executes directly,

security checks are not performed!

General

Source Code Analysis

Before releasing an application and after security-related changes, the complete source code **must** be scanned. We currently use RIPS to perform scans. Affected Software:

- Core
- All apps in core
- All apps in the marketplace

Architecture

Security Related Comments in Source Code

- Security-related comments in source code are forbidden. Source code means PHP code and especially JavaScript code. Security-related comments are:
  - Usernames and passwords
  - Descriptions of processes and algorithms
- Before deploying your code, use a minifier for JavaScript and CSS files.

HTTP or HTTPS

- Only use HTTPS for rendering content.
- Avoid switching between HTTP and HTTPS, which creates mixed-content pages.

Security Related Actions

- All security-related actions must take place on the server. This includes *validation*, *authentication*, and *authorization*. Authorization implementations on the client side are only useful for providing a better user experience.
- Don’t hard-code passwords or encryption keys in the source code. They have to be in config files and should be user-generated.

Browser plugins

Don’t use browser plugins such as:

- ActiveX Controls
- Java Applets
- Flash

Least Privilege Principle

- Every application should only have the rights that it needs.
- An application should not access core database tables. If it needs data from these tables, it should call an API endpoint to retrieve it.

Error Messages and Error Pages

- Don’t show sensitive information on error pages or in error messages. Sensitive information includes:
  - Username/password

- E-Mail addresses
- Version numbers
- Paths

- Don’t show overly detailed information in error messages or on error pages.

  Example:

  If a user can’t login, don’t show an error like: *Your password is wrong*. Instead, show a message such as: *There was an error with your credentials*. If you print *Your password is wrong* then an attacker knows the username was a valid one in the ownCloud installation.

- Consider implementing a CAPTCHA to prevent brute force attacks, after five failed login attempts.

Session ID Transport

- Don’t use a session id as a GET Parameter, because these persist in browser history. Use cookies instead.

New Session ID After a Successful Login

- After a successful login, regenerate the session id to prevent session fixation attacks.
- If you have to switch between HTTPS and HTTP, you should change the session id, because an attacker could have already read the session id.

Access Protection With Authorization Checks

- Every request to the server must check if the user has the authorization to perform this request. We do not recommend running these on the client-side, as they can be avoided. However, client-side checks can improve the user’s experience.

Best Practices

Use of the eval Function

- Don’t use either PHP’s or JavaScript’s eval functions — especially not with user-supplied data.

Input Validation

- All user-supplied data, $_SERVER, and $_COOKIE variables **must** be validated. All these contain data which can be changed (or forged) by the client.
- Sanitize any supplied script code.

Example:

If you expect to receive an integer id as a GET parameter, then always explicitly cast it into an integer using the cast operator (int), because all $_REQUEST parameters are strings. However, if you expect text as a parameter, use PHP’s htmlspecialchars function with ENT_QUOTES or strip_tags to prevent Cross-site Scripting (XSS) attacks.

```
<?php

$neu = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
echo $neu; // &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
```

```php
<?php

$text = '<p>Test-Absatz.</p><!-- Kommentar --> <a href="#fragment">Anderer Text</a>';
echo strip_tags($text);
echo "\n";
```

Output:

```
Test-Absatz. Anderer Text
<p>Test-Absatz.</p> <a href="#fragment">Anderer Text</a>
```

Do the validation **before** all other actions.

Path Traversal and Path Manipulation

- Don't use user-supplied data to build path names, if you need to access the file system. You have to check the input parameters for null bytes (\0), the links to the current and parent directory on UNIX/Linux filesystems (. and ..), and empty strings.

Prevent Command Injection

- Use PHP's escapeshellarg() function, if your input parameters are arguments for exec(), popen(), system(), or the backtick (``) operator.

  ```php
  <?php

  system('ls '.escapeshellarg($dir));
  ```

- If you do not know how many arguments your application receives, then use the PHP function escapeshellcmd() to escape the whole command.

  ```php
  <?php
  $command = './configure '.$_POST['configure_options'];

  $escaped_command = escapeshellcmd($command);

  system($escaped_command);
  ```

Output Escaping

- All input parameters printed out in the response should be escaped.
- Do not use print_unescaped() in ownCloud templates, use p() instead.
- Use $jQuery.text(), if you have to output text in JavaScript .
- Use $jQuery.html(), if you want to output HTML, . A better option is to use a tool like HTMLPurifier.

High Sensitive Information in GET Request

- You should not use sensitive information, like passwords or usernames, in unprotected requests.
- All requests containing sensitive information should be protected with HTTPS.

Prevent HTTP-Header-Injection (HTTP Response Splitting)

- To prevent HTTP Response Splitting, check all request variables for %0d (CR) and %0a (LF), if they are parameters provided to PHP's header() function. This is because an attacker can deface your website, such as redirect the request to a phishing site or executing an XSS attack, by performing header manipulation.

Changes on the Document Object Model (DOM)

Don't use unvalidated user input, if your code changes the DOM.

You should never trust user input.

Prevent SQL-Injection

- Use the escape functions for your database to prevent SQL Injection attacks, if you have to pass parameters to a SQL query. In ownCloud you must use the QueryBuilder.

Data Storage

Persistent Storages on Client Side

- Don't save highly sensitive data in persistent storage on the client side. Persistent data storage includes:
  - Persistent HTTP cookies
  - Flash cookies
  - HTML5 Web-Storage
  - HTML5 Index DB

Release all Resources in Case of an Error

- All resources, such as database and file locks, must be released when errors occur. Doing so prevents the server from being subject to denial-of-service (DOS) attacks.

Cryptography

Symmetric Encryption Methods

- If you use symmetric encryption methods in your code, use the following encryption types:
  - AES with a key length of 256
  - SERPENT with a key length of 256
- For block ciphers use the following modes:
  - CFB (cipher feedback mode)
  - CBC (cipher block chaining mode)

CFB mode requires an initialization vector (IV) to the respective cipher function. Whereas in CBC mode, supplying one is optional. The IV must be unique and must be the same when encrypting and decrypting. Use the PHP crypt library with libmcrypt greater 2.4.x.

Asymmetric Encryption Methods

- If you use asymmetric encryption methods, use RSA encryption with a key length of 4096.

Hash Algorithms

- If you need a hash function in PHP, use the SHA512 hash algorithm.
- You can use PHP's crypt() function, but only with a strong salt.
- Don't use *MD5*, *SHA1* or *SHA256*. These types of algorithms are designed to be very fast and efficient. However, with modern techniques and computer equipment, it has become trivial to brute force the output of these algorithms to discover the original input.

Cookies

Secure Flag

- If you use HTTPS to protect requests, then use the secure flag for your cookies.

HTTP Only

- If you do not have to access your cookie content in JavaScript, then set the HttpOnly flag on every cookie.

Path

- If possible, set a path for a cookie. Doing so ensures that the cookie is only valid for requests using the provided path.

Passwords

The following chapter is not only for developers but also for admins and end-users.

Charset of Passwords

- The charset of a password should contain *characters*, *numbers*, and *special characters*.
- Characters should be both upper and lowercase.

Password Length

- All passwords should have a minimum length of eight characters and contain numbers and special characters. These requirements must be validated by the application.

Password Quality

- If the user can choose his password for the first time, the quality of a password should be displayed graphically.

Password Input

- If a user can input his password into an input field, the input field **must** be of type password.
- If an error occurs, don't fill the password field automatically when displaying an error message.

Save Passwords

- Don’t save passwords in clear text. Use a salted hash

Default and Initial Passwords

- Avoid using both default and initial passwords. If you have to use either, you have to make sure that the password is changed by the user on the first call to the application.

User Interface

Input Auto-completion

- Auto-complete must be disabled for all input fields which receive sensitive data. Sensitive data includes:
  - Username
  - Password
  - Credit card information
  - Banking information
- For text input fields use autocomplete="off" or use a dynamically generated field name.
- For password fields use:

```
<input name="pass" type="password" autocomplete="new-password" />
```

Attack Vectors

Auth bypass / Privilege escalations

Auth bypass/privilege escalations happen when users can perform unauthorized actions. ownCloud offers three simple checks:

- **OCP\JSON::checkLoggedIn()**: Checks if the logged in user is logged in
- **OCP\JSON::checkAdminUser()**: Checks if the logged in user has admin privileges
- **OCP\JSON::checkSubAdminUser()**: Checks if the logged in user has group admin privileges

These checks are already automatically performed, by the application framework, for each request. If they are not required, they have to be *explicitly* turned off by using annotations above your controller method. Additionally, always check /if the user has the right to perform that action.

Clickjacking

Clickjacking tricks the user to click into an invisible iframe to perform an arbitrary action (e.g., deleting a user account).

To prevent such attacks ownCloud sends the X-Frame-Options header to all template responses. Don’t remove this header unless you need to!

This functionality is built into ownCloud when ownCloud templates or Twig Templates are used.

Code executions / File inclusions

Code execution means that an attacker can include an arbitrary PHP file. This PHP file runs with all the privileges granted to the normal application and can do an enormous amount of damage. Code executions and file inclusions can be easily prevented by never allowing user-input to run through the following functions:

- **include()**
- **require()**
- **require_once()**
- **eval()**
- **fopen()**

Never allow the user to upload files into a folder which is reachable from the URL!

DON'T

```
<?php
require("/includes/" . $_GET['file']);
```

If you have to pass user input to a potentially dangerous function, double check to be sure that there is no other option available. If there is no other option, sanitize every user parameter and ask people to audit your sanitize functions.

Cross Site Request Forgery (CSRF)

Using CSRF one can trick a user into executing a request that he did not want to make. Thus every POST and GET request needs to be protected against it. The only places where no CSRF checks are needed are in the main template, which is rendering the application, or in externally callable interfaces.

Submitting a form is also a POST/GET request!

To prevent CSRF in an app, be sure to call the following method at the top of all your files:

```
<?php
OCP\JSON::callCheck();
```

If you are using the application Framework, every controller method is automatically checked for CSRF unless you explicitly exclude it by setting the @NoCSRFRequired annotation before the controller method.

Cross Site Scripting (XSS)

Cross-site scripting happens when user input is passed directly to templates. A potential attacker might be able to inject HTML or JavaScript into the page to steal the user's session, log keyboard entries, or perform DDOS attacks on other websites and other malicious actions.

Despite the fact that ownCloud uses Content-Security-Policy to prevent the execution of inline JavaScript code developers are still required to prevent XSS. CSP is another layer of defense that is not implemented in all web browsers.

To prevent XSS vulnerabilities in your application, you have to sanitize both the templates *and* all JavaScript scripts which perform DOM manipulation.

Templates

Let's assume you use the following example in your application:

```
<?php
echo $_GET['username'];
```

An attacker might now easily send the user a link to app.php?username=<script src="attacker.tld"></script>, to take control of the user account. The same problem occurs when outputting content from the database, or any other location that is writable by users. Another attack vector that is often overlooked is XSS vulnerabilities in href attributes. HTML allows for executing JavaScript in href attributes like this:

```
<a href="javascript:alert('xss')">
```

To prevent XSS in your app, never use echo, print() or <\%=, use p() instead. Doing so sanitizes input. Also **validate URLs to start with the expected protocol** (starts with http for instance)!

Should you ever need to print something unescaped, double check if it is necessary. If there is no other way (e.g., when including sub-templates) use print_unescaped with care.

JavaScript

Avoid manipulating HTML directly via JavaScript. Doing so often leads to XSS vulnerabilities since people often forget to sanitize variables. For example:

```
var html = '<li>' + username + '</li>"';
```

If you want to use JavaScript for something like this use escapeHTML to sanitize the variables:

```
var html = '<li>' + escapeHTML(username) + '</li>';
```

An even better way to make your application safer is to use the jQuery built-in function **$.text()**, instead of **$.html()**.

DON'T

```
messageTd.html(username);
```

DO

```
messageTd.text(username);
```

It may also be wise to choose a proper JavaScript framework, like AngularJS, which automatically handles JavaScript escaping for you.

Directory Traversal

Very often, developers forget about sanitizing the file path (such as removing all \\ and /). Doing so allows an attacker to traverse through directories on the server and opens several potential attack vendors, which include *privilege escalations*, *code executions*, and *file disclosures*.

DON'T

```
<?php
$username = OC_User::getUser();
fopen("/data/" . $username . "/" . $_GET['file'] . ".txt");
```

DO

```
<?php
$username = OC_User::getUser();
$file = str_replace(array('/', '\\'), ,  $_GET['file']);
fopen("/data/" . $username . "/" . $file . ".txt");
```

PHP also interprets the backslash (\) in paths, don't forget to replace it too!

Shell Injection

Shell Injection occurs if PHP code executes shell commands (e.g., running a latex compiler). Before doing this, check if there is a PHP library that already provides the needed functionality. If you really need to execute a command be aware that you have to escape every user parameter passed to one of these functions:

- **exec()**
- **shell_exec()**
- **passthru()**
- **proc_open()**
- **system()**
- **popen()**

Please require/request additional programmers to audit your escape function.

Without escaping the user input, this allows an attacker to execute arbitrary shell commands on your server. PHP offers the following functions to escape user input:

- **escapeshellarg()**: Escape a string to be used as a shell argument
- **escapeshellcmd()**: Escape shell metacharacters

DON'T

```
<?php
system('ls '.$_GET['dir']);
```

DO

```php
<?php
system('ls '.escapeshellarg($_GET['dir']));
```

Sensitive data exposure

Always store user data or configuration files in safe locations, e.g., **owncloud/data/** and not in the web root, where they are accessible by anyone using a web browser.

SQL Injection

SQL Injection occurs when SQL query strings are concatenated with variables. To prevent this, always use prepared queries:

```php
<?php
$sql = 'SELECT * FROM `users` WHERE `id` = ?';
$query = \OCP\DB::prepare($sql);
$params = array(1);
$result = $query->execute($params);
```

If the application Framework is used, write SQL queries like this in the class that extends the Mapper:

```php
<?php
// inside a child mapper class
$sql = 'SELECT * FROM `users` WHERE `id` = ?';
$params = array(1);
$result = $this->execute($sql, $params);
```

Unvalidated redirects

This is more of an annoyance than a critical security vulnerability since it may be used for social engineering or phishing. Before redirecting, always validate the URL if the requested URL is on the same domain or is an allowed resource.

DON'T

```php
<?php
header('Location:'. $_GET['redirectURL']);
```

DO

```php
<?php
header('Location: https://example.com'. $_GET['redirectURL']);
```

Getting Help

If you need help to ensure that a function is secure, please ask on our mailing list or in IRC channel **#owncloud-dev** on **irc.freenode.net**.

Backporting

General

We backport important fixes and improvements from the current master release to get them to our users faster.

Process

We mostly consider bug fixes for back porting. Occasionally, important changes to the API can be backported to make it easier for developers to keep their apps working between major releases. If you think a pull request (PR) is relevant for the stable release, go through these steps:

1. Make sure the PR is merged to master
2. Ask the feature maintainer if the code should be backported and add the label backport-request to the PR
3. If the maintainer say yes then create a new branch based on the respective stable branch, cherry-pick the needed commits to that branch and create a PR on GitHub.
4. Specify the corresponding milestone for that series to this PR and reference the original PR in there. This enables the QA team to find the backported items for testing and having the original PR with detailed description linked.

Before each patch release there is a freeze to be able to test everything as a whole without pulling in new changes. This freeze is announced on the owncloud-devel mailinglist. While this freeze is active a backport isn't allowed and has to wait for the next patch release.

The QA team will try to reproduce all the issues with the X.Y.Z-next-maintenance milestone on the relevant release and verify it is fixed by the patch release (and doesn't cause new problems). Once the patch release is out, the post-fix -next -maintenance is removed and a new -next-maintenance milestone is created for that series.

Backporting Steps

Because pushing directly to particular ownCloud branches is forbidden (e.g., origin/stable-xx), you need to create your own remote branch, based off of the branch that you wish to backport to. However, doing so can involve a number of manual steps. To reduce the effort and time involved, use the script below instead.

Backporting Script

The script uses curl and the jq (lightweight and flexible command-line JSON processor) package. Please install them before first usage. Please see this link for installation details of jq covering varios OS.

This script uses the github API. For unauthenticated requests, the rate limit allows for up to 60 requests per hour. Unauthenticated requests are associated with the originating IP address, and not the user making requests. Please see this link for more information about github rate limiting.

In case of conflicts, the script exits. The merge conflicts will need to be resolved before manually continuing the backport. When done, we suggest that you use the printed subject title from the script for the Pull Request.

```
#!/bin/bash

if ! [ -x "$(command -v jq)" ]; then
  echo 'Error: jq is not installed.' >&2
  echo 'Please install package "jq" before using this script'
  exit 1
fi

if ! [ -x "$(command -v curl)" ]; then
  echo 'Error: curl is not installed.' >&2
  echo 'Please install package "curl" before using this script'
  exit 1
fi

if [ "$#" -lt 2 ]; then
    echo "Illegal number of parameters"
    echo "  $0 <merge/commit-sha> <targetBranchName>"
    echo "  For example: $0 1234567 stable10"
    exit
fi

commit=$1
targetBranch=$2
baseBranch=$(git rev-parse --abbrev-ref HEAD)

is_merged=$(git branch --contains $1 | grep -oP '(?<=\*).*')
if [ -z "$is_merged" ]; then
    echo "$commit has not been merged or branch $baseBranch is not
pulled/rebased. Exiting"
    echo
    exit
fi

# get the PR number from commit
pullId=$(git log $1^! --oneline 2>/dev/null | tail -n 3 | grep -oP '(?<=#)[0-9]*')

# get the repository from commit
repository=$(git config --get remote.origin.url 2>/dev/null | perl -lne 'print $1 if
/(?:(?:https?:\/\/github.com\/)|:)(.*?).git/')

# get the list of commits in PR without any merge commit
# $1^ means the first parent of the merge commit (that is passed in as $1).
# Because $1 is a "magically-generated" merge commit, it happily "jumps back"
# to the point on the main branch just before where the PR was merged.
# And so the commits from that point are exactly the list of individual
# commits in the original PR.
# --no-merges leaves out the merge commit itself, and we get just what we want
commitList=$(git log --no-merges --reverse --format=format:%h $1^..$1)
```

```
# get the request reset time window from github in epoch
rateLimitReset=$(curl -i https://api.github.com/users/octocat 2>&1 | grep -m1 'X-RateLimit-Reset:' | grep -o '[[:digit:]]*')

# get the remaining requests in window from github
rateLimitRemaining=$(curl -i https://api.github.com/users/octocat 2>&1 | grep -m1 'X-RateLimit-Remaining:' | grep -o '[[:digit:]]*')

# time remaining in epoch
now=$(date +%s)
((remaining=rateLimitReset-now))

# time remaining in HMS
remaining=$(date -u -d @$remaining +%H:%M:%S)

# check if there are commits to cherry pick and list them in case
if [[ -z "$commitList" ]]; then
  echo "There are no commits to cherry pick. Exiting"
  exit
else
  lineCount=$(echo "$commitList" | grep '' | wc -l)
  echo "$lineCount commits being cherry picked:"
  echo
  echo "$commitList"
fi

if [ $rateLimitRemaining -le 0 ]; then
  # do not continue if there are no remaining github requests available
  echo "You do not have enough github requests available to backport"
  echo "The current rate limit window resets in $remaining"
  exit
else
  # get the PR title, this is the only automated valid way to get the title
  pullTitle=$(curl https://api.github.com/repos/$repository/pulls/$pullId 2>/dev/null | jq '.title' | sed 's/^.//' | sed 's/.$//')
fi

# build names used
targetCommit="$targetBranch-$commit-$pullId"
message="[$targetBranch] [PR $pullId] $pullTitle"

echo
echo "Info:"
echo "You have $rateLimitRemaining backport requests remaining in the current github rate limit window"
echo "The current rate limit window resets in $remaining"
echo
echo "Backporting commit $commit to $targetBranch with the following text:"
echo "$message"
echo
```

```
set -e

git fetch -p --quiet
git checkout "$targetBranch"
git pull --rebase --quiet
git checkout -b "$targetCommit"

echo

# cherry pick all commits from commitList
lC=1
echo "$commitList" | while IFS= read -r line; do
  echo "Cherry picking commit $lC: $line"
  # if you do not want to use a default conflict resolution to take theirs
  # (help fix missing cherry picked commits or file renames)
  #git cherry-pick $line > /dev/null
  git cherry-pick -Xtheirs $line > /dev/null
  lC=$(( $lC + 1 ))
done
echo

git commit --quiet --amend -m "$message" -m "Backport of PR #$pullId"

echo "Pushing: $message"
echo

git push --quiet -u origin "$targetCommit"
git checkout --quiet "$baseBranch"
```

It is highly recommended to use the merge commit sha when backporting a Pull Request. The merge commit includes all PR sub commits to be backported. With that, no individual sub commit backporting is necessary.

The following example assumes that:

- You save the script in a file called <path>/backport.sh and mark it executable
- You have checked out the branch containing the merged sha (like master)
- Your Pull Request merge sha = 1234567 and your target branch = stable10

The command to backport this Pull Request would be called as follows:

```
<path>/backport.sh 1234567 stable10
4 commits beeing cherry picked:

2e03d938
fef19729
61ac3f09
0528601f
...
Switched to a new branch ‘stable10-1234567-34654‘
...
[stable10] [PR 34654] Each generated birthday or death event gets a new UID
...
Cherry picking commit 1: 2e03d938
Cherry picking commit 1: fef19729
...
Pushing: ...
...
```

Please keep in mind that this is an example and you have to adapt the commit hash and the target branch accordingly.

The script lists the quantity and commits to be backported and the current one in process. This can be helpful in case there is a conflict and you manually continue after the conflict has been resolved.

When the script completes, go to GitHub, where it will suggest that you make a PR from pushed branch. Even the script tries to automate, you may need to set the Pull Request subject and message text manually via copy/paste.

Change the base branch to be committed against, from master to your target branch (in our example stable10) and continue.

In case you have installed the xdg-utils package, you can add at the end of the script above following code which opens the PR to be finalized in your browser. macOS does not need this package, use the command open instead:

```
echo "Creating pull request for branch $targetBranch in $repository"

xdg-open "https://github.com/$repository/pull/new/$targetBranch...$targetCommit"
&>/dev/null
```

This command opens the Pull Request and sets the target branch (in our example stable10) for the backport automatically.

Backporting Alias

Open the ~/.gitconfig file with the editor of your choice and add the following:

```
[alias]
   backport = !bash -c '<path_to_script>/backport.sh $1 $2' -
```

Create a backport by invoking following command

```
git backport 1234567 stable10
```

Please keep in mind that this is an example and you have to adapt the commit hash and the target branch accordingly.

Help and Communication

Getting Involved

Introduction

There are a variety of ways to get involved and seek help if and when you need it. Here's the best ways.

Mailing Lists

On the mailing lists.

Community Support Forum

Ask questions on ownCloud Central. We strongly recommend using ownCloud Central, as it hosts dedicated FAQ pages. These include topics which address typical mistakes and commonly occurring issues.

Social Media

Ask questions on social media:

- Facebook
- Twitter

IRC Channels

Chat with us on IRC (**irc.freenode.net**). You can chat via the web with http://webchat.freenode.net, or use your favorite IRC client. The channel names are:

- Setup: **#owncloud**
- Testing: **#owncloud-testing**
- Development: **#owncloud-dev**
- Design: **#owncloud-design**

Maintainers

If you need to contact a maintainer of a certain app or division you can find the details at https://owncloud.org/contact/.

Core Development

In this section you will find all the details you need to develop ownCloud's core.

Introduction

Please make sure you have set up a Development Environment.

Setup Your Development Environment

Introduction

This page helps with setting up your environment for use with and developing ownCloud.

Feel free to skip already one or more of the following steps, if you have already completed them. Otherwise, if you're just getting started, begin by getting the ownCloud source code.

Install the Core Software

The first thing to do is to ensure that your server has the necessary software for installing and running ownCloud. While you can go further, you need to install at least the required packages. Then, you will need to install the software required to run the development environment's installation process. These are: GNU make, Node.js, git, npm, unzip, wget

Install Dependencies on Ubuntu 16.04/18.04

Install nodejs, make, unzip, and git

```
cd ~
curl -sL https://deb.nodesource.com/setup_8.x -o nodesource_setup.sh
sudo bash nodesource_setup.sh
sudo apt-get -y -q update
sudo apt-get -y -q upgrade
sudo apt-get install nodejs build-essential make unzip git
```

Install Composer

Prepare the Installation

```
cd ~/tmp
sudo apt-get install wget php-cli php-zip
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
HASH="$(wget -q -O - https://composer.github.io/installer.sig)"
php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
```

If the hashes match, you will see the following output:

```
Installer verified
```

Install Composer

To install Composer, run the following command:

```
sudo php composer-setup.php \
    --install-dir=/usr/local/bin \
    --filename=composer
```

Running the command will produce output similar to the following.

```
All settings correct for using Composer
Downloading...

Composer (version 1.7.2) successfully installed to: /usr/local/bin/composer
Use it: php /usr/local/bin/composer
```

Verify the Installation

To verify that Composer is properly installed, run composer. You should see output similar to that below.

```
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.8.4 2019-02-11 10:52:10
```

Composer is fully installed, and ready to be used.

Install Yarn

```
# Enable the Yarn repository
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -

# Add the Yarn APT repository to your system's software repository list:
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list

# Update the package list and install Yarn:
sudo apt-get update
sudo apt-get install --no-install-recommends yarn
```

Verify that Yarn installed successfully:

After you have installed yarn, you can run yarn --version to confirm that it's fully installed. If it is, then it will print output similar to the following to the console.

```
yarn version v1.13.0
```

Install Dependencies on openSUSE Leap 42.3

```
# Ensure that Zypper's cache is up to date
sudo zypper --non-interactive --quiet \
  update --auto-agree-with-licenses --best-effort

# Auto-install the required dependencies with a minimum of output
sudo zypper --quiet --non-interactive install \
    wget make nodejs6 nodejs-common unzip git
    npm6 phantomjs php7-curl php7-openssl openssl php7-phar
```

Setup the Webserver and Database

Next, you need to setup your web and database servers, so that they work properly with ownCloud. The respective guides are available at:

- Apache Webserver Configuration
- Database Server Configuration

Get The Source

With the web and database servers setup, you next need to get a copy of ownCloud. There are two ways to do so:

1. Use a manual installation
2. Use a Linux Package Manager Installation
3. Clone the development version from GitHub:

For the sake of a brief example, assuming you chose to clone from GitHub, here's an example of how to do so:

```
# Assuming that /var/www/html is the webserver's document root
git clone https://github.com/owncloud/core.git /var/www/html/core
```

What is the Web Server's Root Directory?

The quickest way to find out is by using the ls command, for example: ls -lah /var/wwww. Depending on your Linux distribution, it's likely to be one of /var/www, /var/www/html, or /srv/http.

Set User, Group, and Permissions

You now need to make sure that the web server user (and optionally the web server's group) have read/write access to the directory where you installed ownCloud: The following commands assume that /var/www is the web server's directory and that www-data is the web server user and group. The following commands will do this:

```
# Set the user and group to the webserver user and group
sudo chown -R www-data:www-data /var/www/html/core/

# Set read/write permissions on the directory
sudo chmod o+rw /var/www/html/core/
```

What is the Web Server's User and Group?

There are a few ways to identify the user and group the webserver is running as. Likely the easiest are grep and ps. Here's an example of using both (which assumes that the distribution is Ubuntu 16.04).

```
# Find the user defined in Apache's configuration files
grep -r 'APACHE_RUN_USER' /etc/apache2/

# Find the user that's running Apache.
ps -aux | grep apache2
```

Depending on your distribution, it will likely be one of http, www-data, apache, or wwwrun.

Install Software Dependencies

With the ownCloud source available to your webserver, next install ownCloud's dependencies by running Make, from the directory where ownCloud's located. Here's an example of how to do so:

```
# Assuming that the ownCloud source is located in `/var/www/html/core`
cd /var/www/html/core && make
```

By default, running make will install the required dependencies for both PHP and JavaScript. However, there are other options that it supports, which you can see in the table below, which are useful for a variety of tasks.

Target	Description
make	Pulls in both Composer and Bower dependencies
make clean	Cleans up dependencies. This is useful for starting over or when switching to older branches
make dist	Builds a minimal owncloud-core tarball with only core apps in build/dist/core, stripped of unwanted files
make docs	Builds the JavaScript documentation using JSDoc
make test	Runs all of the test targets
make test-external	Runs one of the external storage tests, and is configurable through make variables
make test-js	Runs the Javascript unit tests, replacing ./autotest-js.sh
make test-php	Runs the PHPUnit tests with SQLite as the data source. This replaces ./autotest.sh sqlite and is configurable through make variables

Enable Debug Mode

Now that ownCloud's available to your web server and the dependencies are installed, we strongly encourage you to disable JavaScript and CSS caching during development. This is so that when changes are made, they're immediately visible, not at some later stage when the respective caches expire. To do so, enable debug mode by setting debug to true in config/config.php, as in the example below.

```
<?php

$CONFIG = [
   'debug' => true,
   ... configuration goes here ...
];
```

Do not enable this for production! This can create security problems and is only meant for debugging and development!

Setup ownCloud

With all that done, you're now ready to use either the installation wizard or command line installer to finish setting up ownCloud.

Application Configuration

```
<?php

$CONFIG = [
 /* Flag to indicate ownCloud is successfully installed (true = installed) */
 'installed' => false,

 /* Type of database, can be sqlite, mysql or pgsql */
 'dbtype' => 'sqlite',

 /* Name of the ownCloud database */
 'dbname' => 'owncloud',

 /* User to access the ownCloud database */
 'dbuser' => '',

 /* Password to access the ownCloud database */
 'dbpassword' => '',

 /* Host running the ownCloud database */
 'dbhost' => '',

 /* Prefix for the ownCloud tables in the database */
 'dbtableprefix' => '',

 /**
  Define the salt used to hash the user passwords.
  All your user passwords are lost if you lose this string.
```

```
 */
'passwordsalt' => '',

/* Force use of HTTPS connection (true = use HTTPS) */
'forcessl' => false,

/* Theme to use for ownCloud */
'theme' => '',

/* Path to the 3rdparty directory */
'3rdpartyroot' => '',

/* URL to the 3rdparty directory, as seen by the browser */
'3rdpartyurl' => '',

/* Default app to load on login */
'defaultapp' => 'files',

/* Enable the help menu item in the settings */
'knowledgebaseenabled' => true,

/* Enable installing apps from the appstore */
'appstoreenabled' => true,

/* URL of the appstore to use, server should understand OCS */
'appstoreurl' => 'https://api.owncloud.com/v1',

/* Mode to use for sending mail, can be sendmail, smtp, qmail or php, see
PHPMailer docs */
'mail_smtpmode' => 'sendmail',

/* Host to use for sending mail, depends on mail_smtpmode if this is used */
'mail_smtphost' => '127.0.0.1',

/* authentication needed to send mail, depends on mail_smtpmode if this is used
 * (false = disable authentication)
 */
'mail_smtpauth' => false,

/* Username to use for sendmail mail, depends on mail_smtpauth if this is used */
'mail_smtpname' => '',

/* Password to use for sendmail mail, depends on mail_smtpauth if this is used */
'mail_smtppassword' => '',

/* Check 3rdparty apps for malicious code fragments */
'appcodechecker' => '',

/* Check if ownCloud is up to date */
'updatechecker' => true,
```

```
  /* Place to log to, can be owncloud and syslog (owncloud is log menu item in
admin menu) */
  'log_type' => 'owncloud',

  /* File for the owncloud logger to log to, (default is ownloud.log in the data dir */
  'logfile' => '',

  /* Loglevel to start logging at. 0=DEBUG, 1=INFO, 2=WARN, 3=ERROR (default is
WARN) */
  'loglevel' => '',

  /* Lifetime of the remember login cookie, default is 15 days */
  'remember_login_cookie_lifetime' => 60*60*24*15,

  /* The directory where the user data is stored, default to data in the owncloud
   * directory. The sqlite database is also stored here, when sqlite is used.
   */
  'datadirectory' => '/var/www/owncloud/data',

  /* Set an array of path for your apps directories
   key 'path' is for the filesystem path and the key 'url' is for the http path to your
   applications paths. 'writable' indicates if the user can install apps in this folder.
   You must have at least 1 app folder writable or you must set the parameter :
appstoreenabled to false.
  */
  'apps_paths' => [
     [
        'path' => OC::$SERVERROOT.'/apps',
        'url' => '/apps',
        'writable' => true,
     ],
  ]
]
```

Using alternative app directories

ownCloud can be set to use a custom app directory in /config/config.php. Customise the following code and add it to your config file:

```
'apps_paths' => [
   [
      'path' => OC::$SERVERROOT.'/apps',
      'url' => '/apps',
      'writable' => false,
   ],
   [
      'path' => OC::$SERVERROOT.'/apps-external',
      'url' => '/apps-external',
      'writable' => true,
   ],
],
```

ownCloud will use for new or app updates the first app directory which it finds in the array with writable set to true.

Theming ownCloud

Introduction

Themes can be used to customize the look and feel of any aspect of an ownCloud installation. They can override the default *JavaScript*, *CSS*, *image*, and *template* files, as well as the *user interface translations* with custom versions. They can also affect both the web front-end and the ownCloud Desktop client. However, this documentation only covers customizing the web front-end, *so far*.

Before ownCloud 10, theming was done via the config.php entry 'theme' ⇒ ''. This is deprecated in ownCloud 10. Users who have this entry in their config.php should remove it and use a theme app to customize their ownCloud instance instead.

Throughout this section of the documentation, for sakes of simplicity, it will be assumed that your owncloud installation directory is /owncloud. If you're following this guide to create or customise a theme, please make sure you change any references to match the location of your owncloud installation.

To save you time and effort, you can use the shell script below, to create the basis of a new theme from ownCloud's example theme.

Using this script (and the following one, read-config.php), you will have a new theme, ready to go, in less than five seconds. You can execute this script with two variables; the first one is the **theme name** and the second one is your **ownCloud directory**.

For example:

```
theme-bootstrap.sh mynewtheme /var/www/owncloud
```

Don't forget to create read-config.php from the included code below, before you attempt to run theme-bootstrap.sh, otherwise theme-bootstrap.sh will fail.

theme-bootstrap.sh

```
#!/bin/bash
# theme-bootstrap.sh
# Invoke this script with two arguments, the new theme's name and the path to
ownCloud root.
# Written by Dmitry Mayorov <dmitry@owncloud.com>,  Matthew Setter
<matthew@matthewsetter.com> & Martin Mattel <github@diemattels.at>
# Copyright (c) ownCloud 2018.
set -e

# Clone a copy of the ownCloud example theme
# It won't override an existing app directory of the same name.
function clone_example_theme
{
  local APP_NAME="$1"
  local INSTALL_BASE_DIR="$2"
  local MAINFILE=master.zip
  local UNZIPDIR=/tmp
  local MASTERNAME=theme-example-master
  local DOWNLOAD_FILE=$UNZIPDIR/$MAINFILE
  local THEME_ARCHIVE_URL=https://github.com/owncloud/theme-
example/archive/master.zip

  # check if the app name already exists
  if  [ -d "$INSTALL_BASE_DIR/$APP_NAME" ]
  then
    echo "An app with name ('$INSTALL_BASE_DIR/$APP_NAME') already exists."
    echo "Please remove or rename it before running this script again."
    return 1
  fi;

  # delete an existing downloaded zip file
  if [ -e "$DOWNLOAD_FILE" ]
  then
    rm "$DOWNLOAD_FILE"
  fi

  echo "Downloading ownCloud example theme."

  # getting the exmple theme from git
  if ! wget --output-document="$DOWNLOAD_FILE" --tries=3 --continue \
    --timeout=3 --dns-timeout=3 --connect-timeout=3 --read-timeout=3  \
    "$THEME_ARCHIVE_URL" >/dev/null 2>&1
  then
    echo "Download error, exiting"
    return 1
  fi

  # first test if unzip would error then extract
  if unzip -t "$DOWNLOAD_FILE" >/dev/null 2>&1
  then
```

```
    # unzip with overwriting existing files and directories and suppressed output
    echo "Unzipping download"
    unzip -oq "$DOWNLOAD_FILE" -d "$UNZIPDIR"
    echo "Moving to target location"
    mv "$UNZIPDIR/$MASTERNAME" "$INSTALL_BASE_DIR/$APP_NAME"
    echo "Removing download"
    rm "$DOWNLOAD_FILE"
  else
    echo "Cannot complete setup of the ownCloud example theme as it is
corrupted."
    return 1
  fi
}
E_BADARGS=85

# Remembers the directory where this script was called from
SCRIPT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null && pwd )"

# Check if run as sudo (root), needed for sub script calling and changing file
permissions
if (( $EUID != 0 )); then
    echo "Please run this script with sudo or as root"
    exit
fi

# Check if enough parameters have been applied
if (( $# != 2 ))
then
  echo "Not enough arguments provided."
  echo "Usage: $( basename "$0" ) [new theme name] [owncloud root directory]"
  exit $E_BADARGS
fi

# Check if read-config.php file exists in the same directory
if [ ! -f $SCRIPT_DIR/read-config.php ]
then
    echo "File read-config.php not found! Must be in the same dir as this script"
    exit
fi

# Check if php file is set to be executable, script will else not work
if [ ! -x $SCRIPT_DIR/read-config.php ]
then
    echo "File read-config.php is not set executable"
    exit
fi

app_name="$1"
owncloud_root="$2"
apps=$(php "$SCRIPT_DIR/read-config.php" "$owncloud_root")
```

```
# Check if the php script returned an error message. This is when the string does
not start with /
if [[ ! $apps = '/'* ]]
then
    echo $apps
    echo "Script read-config.php returned no usable app path"
    exit
fi

if clone_example_theme "$app_name" "$apps"
then
  # Remove the default signature, which will cause a code integrity violation
  [ -f "$apps/$app_name/appinfo/signature.json" ] && rm "$apps/
$app_name/appinfo/signature.json"

  # Replace the default theme id / theme name
  echo "Updating theme id / theme name"
  sed -i "s#<id>theme-example<#<id>$app_name<#" "$apps/
$app_name/appinfo/info.xml"

  # Set the appropriate permissions
  echo "Setting new theme file permissions"
  chown -R www-data:www-data "$apps/$app_name"

  # Enable the new theme app
  if [ -e "$owncloud_root/occ" ]
  then
    echo "Enabling new theme in ownCloud"
    php "$owncloud_root/occ" app:enable "$app_name"
  else
    echo
    echo "occ command not found, please enable the app manually"
  fi

  echo
  echo "Finished bootstrapping the new theme."
fi
```

read-config.php

```
#!/usr/bin/php
<?php
/**
 * @author Matthew Setter <matthew@matthewsetter.com> & Martin Mattel
<github@diemattels.at>
 * @copyright Copyright (c) 2018, ownCloud GmbH
 * @license AGPL-3.0
 *
 * This code is free software: you can redistribute it and/or modify
```

```
 * it under the terms of the GNU Affero General Public License, version 3,
 * as published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License,
version 3,
 * along with this program.  If not, see <http://www.gnu.org/licenses/>
 *
 */

/**
 * Class SimpleConfigReader
 * @package ConfigReader
 */
class SimpleConfigReader
{
    /**
     * @var array
     */
    private $config = [];

    /**
     * String returned to the user.
     * @var string
     */
    private $output = '';

    /**
     * SimpleConfigReader constructor.
     * @param string $config
     */
    public function __construct($config = '')
    {
        $this->config = $config;
    }

    /**
     * Find a writable app directory path that is either defined by key 'apps_paths'
     * or use the default owncloud_root/apps path if the key is not set
     *
     * @return string
     * @throws \Exception
     */
    function findPath($ocAppsPath)    {

        // default path = /apps
```

```
        if (!array_key_exists('apps_paths', $this->config)) {
            $this->output = $ocAppsPath;
            return $this->output;
        }

        foreach ($this->config['apps_paths'] as $path) {
            if ($path['writable'] == true && is_writable($path['path'])) {
                $this->output = $path['path'];
                return $this->output;
            }
        }
        return "Key 'apps_paths' found, but no writable path defined or path found
not writeable";
    }
}

/*
 * As per the PHP manual: The first argument $argv[0] is always the name that
 * was used to run the script. So we need at least two to access the new app's
 * name, as well as the running script's name.
 * @see https://secure.php.net/manual/en/reserved.variables.argv.php
 */
if (count($argv) != 2) {
    echo "Command usage: read-config.php <full path to ownCloud root dir> \n";
    echo "Please provide the path to the ownCloud directory. \n";
    exit(1);
}

// create a realpath and remove trailing "/" from argument if present
$ocRoot = rtrim( (string) $argv[1], "/");
$ownCloudConfigFile = sprintf("%s/config/config.php", $ocRoot);

if (!realpath($ownCloudConfigFile)) {
    // if path/file does not exist, return an error message
    echo 'File not found: ' . $ownCloudConfigFile . PHP_EOL;
} else {
    // return the path, identified by a leading "/" and no new line character at the end
    require_once($ownCloudConfigFile);
    $result = (new SimpleConfigReader($CONFIG))->findPath($ocRoot . '/apps');
    if (!strpos($result, '/')) {
        // return an error string which does not start with a leading "/"
        echo $result  . PHP_EOL;
    } else {
        // return the path, identified by a leading "/" and no new line character at the
end
        echo $result;
    }
}
```

How to Create a New Theme

At its most basic, to create a theme requires two steps:

1. Copy and extend ownCloud's example theme, or create one from scratch.
2. Enable the theme in the ownCloud Admin dashboard.

All themes, whether copied or new, must meet two key criteria, these are:

1. They must be store in an app directory of your ownCloud installation, whether that's the core app directory (apps) or a custom app directory.
2. They require a configuration file called appinfo/info.xml to be present.

To ensure that custom themes aren't lost during upgrades, we strongly encourage you to store them in a custom app directory.

appinfo/info.xml

Here's an example of the bare minimum which the file needs to contain:

```
<?xml version="1.0"?>
<info>
   <id>theme-example</id>
   <name>Example Theme</name>
   <types>
      <theme/>
   </types>
   <dependencies>
    <owncloud min-version="10" max-version="10" />
   </dependencies>
</info>
```

And here's a longer, more complete example:

```
<?xml version="1.0"?>
<info>
   <id>theme-example</id>
   <name>Example Theme</name>
   <description>This App provides the ownCloud theme.</description>
   <licence>AGPL</licence>
   <author>John Doe</author>
   <version>0.0.1</version>
   <types>
      <theme/>
   </types>
   <dependencies>
      <owncloud min-version="10" max-version="10" />
   </dependencies>
</info>
```

The value of the id element needs to be the name of your theme's folder. We recommend that it always be prefixed with theme-. The main reason for doing so, is

that it is alphabetically sorted in a terminal when handling app folders.

The type element needs to be the same as is listed above, so that ownCloud knows to handle the app as a theme. The dependencies element needs to be present to set the minimum and maximum versions of ownCloud which are supported. If it's not present, a warning will be displayed in ownCloud 10 and an error will be thrown in the upcoming ownCloud 11.

While the remaining elements are optional, they help when working with the theme in the ownCloud Admin dashboard. Please consider filling out as many as possible, as completely as possible.

Theme Signing

If you are going to publish the theme as an app in the marketplace, you need to sign it. However, if you are only creating a private theme for your own ownCloud installation, then you do not need to.

That said, to avoid a signature warning in the ownCloud UI, you need to add it to the integrity.ignore.missing.app.signature list in config/config.php. The following example allows the app whose application id is app-id to have no signature.

```
'integrity.ignore.missing.app.signature' => [
    'app-id',
],
```

How to Override Images

Any image, such as the default logo, can be overridden by including one with the same path structure in your theme. For example, let's say that you want to replace the logo on the login-page above the credentials-box which, by default has the path: owncloud/core/img/logo-icon.svg. To override it, assuming that your custom theme was called theme-example (*which will be assumed for the remainder of the theming documentation*), add a new file with the following path: owncloud/apps/theme-example/core/img/logo-icon.svg. After the theme is activated, this image will override the default one.

Default Image Paths

To make building a new theme that much easier, below is a list of a range of the image paths used in the default theme.

Description	Section	Location
The logo at the login-page above the credentials-box	General	owncloud/core/img/logo-icon.svg
The logo in the left upper corner after login		owncloud/core/img/logo-icon.svg
All files folder image		owncloud/core/img/folder.svg
Favorites star image		owncloud/core/img/star.svg
Shared with you/others image		owncloud/core/img/shared.svg
Shared by link image		owncloud/core/img/public.svg

Description	Section	Location
Tags image		owncloud/core/img/tag.svg
Deleted files image		owncloud/core/img/delete.svg
Settings image		owncloud/core/img/actions/settings.svg
Search image		owncloud/core/img/actions/search-white.svg
Breadcrumbs home image		owncloud/core/img/places/home.svg
Breadcrumbs separator		owncloud/core/img/breadcrumb.svg
Dropdown arrow	Admin Menu	owncloud/core/img/actions/caret.svg
Personal image		owncloud/settings/img/personal.svg
Users image		owncloud/settings/img/users.svg
Help image		owncloud/settings/img/help.svg
Admin image		owncloud/settings/img/admin.svg
Logout image		owncloud/core/img/actions/logout.svg
Apps menu - Files image		owncloud/apps/files/img/app.svg
Apps menu - Plus image		owncloud/settings/img/apps.svg
Upload image	Personal	owncloud/core/img/actions/upload.svg
Folder image		owncloud/core/img/filetypes/folder.svg
Trash can image		owncloud/core/img/actions/delete.svg

When overriding the favicon, make sure your custom theme includes and override for both owncloud/apps/core/img/favicon.svg and owncloud/apps/core/img/favicon.png, to cover any future updates to favicon handling.

When using custom filetype icons in a custom theme, it is necessary to run occ maintenance:mimetype:update-js to activate them. For more information please refer to mimetypes management.

How to Override the Default Colors

To override the default style sheet, create a new CSS style sheet in your theme, in the theme's css directory, called styles.css.

How to Override Translations

You can override the translation of any string in your theme. To do so:

1. Create the l10n folder inside your theme, for the app that you want to override.
2. In the l10n folder, create the translation file for the language that you want to customize.

For example, if you want to overwrite the German translation of `*Download*` in the files app, you would create the file owncloud/apps/theme-example/apps/files/l10n/de_DE.js. Note that the structure is the same as for images. You just mimic the original file location inside your theme. You would then put the following code in the file:

```
OC.L10N.register(
  "files",
  {
    "Download" : "Herunterladen"
  },
  "nplurals=2; plural=(n != 1);"
);
```

You then need to create a second translation file, owncloud/apps/theme-example/apps/files/l10n/de_DE.json, which looks like this:

```
{
  "translations": {
    "Download" : "Herunterladen"
  },
  "pluralForm" :"nplurals=2; plural=(n != 1);"
}
```

Both files (.js and .json) are needed. The first is needed to enable translations in the JavaScript code and the second one is read by the PHP code and provides the data for translated terms.

How to Override Names, Slogans, and URLs

In addition to translations, the ownCloud theme allows a lot of the names that are shown on the web interface to be changed. This is done in defaults.php, which needs to be located within the theme's root folder. You can find a sample version in owncloud/app/theme-example/defaults.php. In there, you need to define a class named OC_Theme and implement the methods that you want to overwrite.

```
class OC_Theme {
  public function getAndroidClientUrl() {
    return 'https://play.google.com/store/apps/details?id=com.owncloud.android';
  }

  public function getName() {
    return 'ownCloud';
  }
}
```

Each method must return a string. The following methods are available:

Method	Description
getAndroidClientUrl	Returns the URL to Google Play for the Android Client.
getBaseUrl	Returns the base URL.
getDocBaseUrl	Returns the documentation URL.
getEntity	Returns the entity (e.g., company name) used in footers and copyright notices.
getName	Returns the short name of the software.
getHTMLName	Returns the short name of the software containing HTML strings.
getiOSClientUrl	Returns the URL to the ownCloud Marketplace for the iOS Client.
getiTunesAppId	Returns the AppId for the ownCloud Marketplace for the iOS Client.
getLogoClaim	Returns the logo claim.
getLongFooter	Returns the long version of the footer.
getMailHeaderColor	Returns the mail header color.
getSyncClientUrl	Returns the URL where the sync clients are listed.
getTitle	Returns the title.
getShortFooter	Returns short version of the footer.
getSlogan	Returns the slogan.

Only these methods are available in the templates, because we internally wrap around hardcoded method names.

One exception is the method buildDocLinkToKey which gets passed in a key as its first parameter. For core we do something like this to build the documentation

```
public function buildDocLinkToKey($key) {
  return $this->getDocBaseUrl() . '/server/latest/go.php?to=' . $key;
}
```

How to Test a Theme

There are different options for testing themes:

- If you're using a tool like the Inspector tools inside Mozilla you can test out the CSS-Styles immediately inside the css-attributes, while you're looking at the page.
- If you have a development server, you can test out the effects in a live environment.

Settings Page Registration

How Can an App Register a Section in the Admin or Personal Section?

As of ownCloud 10.0, apps must register admin and personal section settings in info.xml. As a result, all calls to OC_App::registerPersonal and OC_App::registerAdmin should now be removed. The settings panels of any apps that are still using these calls will now be rendered in the Additional section of the dashboard .

For each panel an app wishes to register, two things are required:

1. An update to info.xml
2. A controller class

Updating info.xml

First, an entry must be added into the <settings> element in info.xml, specifying the class name responsible for rendering the panel. These will be loaded automatically when an app is enabled. For example, to register an admin and a personal section would require the following configuration..

```
<settings>
    <personal>OCA\MyApp\PersonalPanel::class</personal>
    <admin>OCA\MyApp\AdminPanel::class</admin>
</settings>
```

The Controller Class

Next, a controller class which implements the OCP\Settings\ISettings interface must be created to represent the panel. Doing so enforces that the necessary settings panel information is returned. The interface specifies three methods:

- getSectionID
- getPanel
- getPriority

getSectionID: This method returns the identifier of the section that this panel should be shown under. ownCloud Server comes with a predefined list of sections which group related settings together; the intention of which is to improve the user experience. This can be found here in this example:

getPanel: This method returns the OCP\Template or OCP\TemplateReponse which is used to render the panel. The method may also return null if the panel should not be shown to the user.

getPriority: An integer between 0 and 100 representing the importance of the panel (higher is more important). Most apps should return a value:

- between 20 and 50 for general information.

- greater than 50 for security information and notices.
- lower than 20 for tips and debug output.

Here's an example implementation of a controller class for creating a personal panel in the security section.

```
<?php

namespace OCA\YourApp

use OCP\Settings\ISettings;
use OCP\Template;

class PersonalPanel extends ISettings {

    const PRIORITY = 10;

    public function getSectionID() {
        return 'security';
    }

    public function getPriority() {
        return self::PRIORITY;
    }

    public function getPanel() {
        // Set the template and assign a template variable
        return (new Template('app-name', 'template-name'))->assign('var', 'value');
    }
}
```

Create Custom Sections

At the moment, there is no provision for apps creating their own settings sections. This is to encourage sensible and intelligent grouping of the settings panels which in turn should improve the overall user experience. If you think a new section should be added to core however, please create a PR with the appropriate changes to OC\Settings\SettingsManager.

Translation

Make text translatable

In HTML or PHP wrap it like this <?php p($l→t('This is some text'));?> or this <?php print_unescaped($l→t('This is some text'));?> For the right date format use <?php p($l→l('date', time()));?>. Change the way dates are shown by editing /core/l10n/l10n-[lang].php To translate text in javascript use: t('appname','text to translate');

print_unescaped() should be preferred only if you would like to display HTML code. Otherwise, using p() is strongly preferred to escape HTML characters against XSS attacks.

You shall never split sentences!

Reason:

Translators lose the context and they have no chance to possibly re-arrange words.

Example:

```
<?php p($l->t('Select file from')) . ' '; ?><a href='#' id="browselink"><?php p($l
->t('local filesystem'));?></a><?php p($l->t(' or ')); ?><a href='#' id=
"cloudlink"><?php p($l->t('cloud'));?></a>
```

Translators will translate:

- Select file from
- local filesystem
- ’ or "
- cloud

Translating these individual strings results in local filesystem and cloud losing case. The two white spaces surrounding or will get lost while translating as well. For languages that have a different grammatical order it prevents the translators from reordering the sentence components.

Html in translation strings

Html tags can be kept out of translation strings like in the example below. Then the detail of the tags is uncoupled from the translation.

What about variables in the strings?

If you need to add variables to the translation strings do it like this:

```
$l->t('%1$s is available. Get %2$smore information%3$s', [$data['versionstring'],
'<a href="' . $data['web']] . '">', '</a>');
```

When there are multiple substitutions, number them. Then the translators have the chance to re-order them if they need to translate the whole sentence in a different word order.

Automated synchronization of translations

Multiple nightly jobs have been setup in order to synchronize translations - it’s a multi-step process: perl l10n.pl read will rescan all php and javascript files and generate the templates. The templates are pushed to Transifex (tx push -s). All translations are pulled from Transifex (tx pull -a). perl l10n.pl write will write the php files containing the translations. Finally the changes are pushed to git.

Please follow the steps below to add translation support to your app:

Create a folder l10n. Create the file ignorelist which can contain files which shall not be scanned during step 4. Edit l10n/.tx/config and copy/past a config section and adopt it by changing the app/folder name. Run perl l10n.pl read with l10n Add the newly created translation template (l10n/Templates/<appname>.pot) to git and commit the changes above. After the next nightly sync job a new resource will appear on Transifex and from now on every night the latest translations will arrive.

Translation sync jobs:

https://drone.owncloud.com/owncloud/translation-sync

Caution: information below is in general not needed!

Manual quick translation update:

```
cd l10n/ && perl l10n.pl read && tx push -s && tx pull -a && perl l10n.pl write &&
cd ..
```

The translation script requires Locale::PO, installable via apt-get install liblocale-po-perl

Configure transifex

```
tx init

for resource in calendar contacts core files media settings
do
tx set --auto-local -r owncloud.$resource "<lang>/$resource.po" --source
-language=en \
 --source-file "templates/$resource.pot" --execute
done
```

Unresolved directive in modules/developer_manual/pages/testing/index.adoc - include::admin_manual:/drone/src/modules/developer_manual/pages/_partials/section_page.adoc[optional attributes]

ownCloud Test Pilots

Introduction

The ownCloud Test Pilots help to test and improve different server and client setups with ownCloud.

What do you do

You will receive emails from the mailing list and also from the bug tracker if developers need your help. Also, there will be announcements of new releases and preview releases on the mailing list, which give you the possibility to test releases early and to help the developers fix them.

We are looking forward to working with you :)

Why do you want to join

There are many different setups, and people have different interests. If we want ownCloud to run well on a variety of different software configurations, someone has to test them. Furthermore, during bug fixing the ownCloud developers often do not have the possibility to reproduce the bug in a given environment, nor are they able confirm if it was fixed.

As a member of the Test Pilot Team you could act as a contact person for a particular area to help developers **fix the bugs you care about**. Testing ownCloud before it is released is the best way of making sure it does what you need.

Another benefit is a closer relationship with the developers, because you will know which people are responsible for which parts, and it will be easier to get help.

If you want, you can also be listed as an active contributor on the owncloud.org page.

Who can join

Anyone who is interested in improving the quality on his/her setup and is willing to communicate with developers and other testers.

How do you join

Just register on the testpilot mailing list and send an introduction containing your personal setup and interests to testpilots@owncloud.org. For further questions or help you can also send a mail to mstingl@owncloud.com.

How do you test

Testing follows these steps:

1. Setup your testing environment
2. Pick something to test
3. Test it
4. Go Back to step 2 until something unexpected/bad happens
5. Check if what you found is a genuine bug
6. File the bug

Installing ownCloud

Testing starts with setting up a testing environment. We urge you not to put your production data on testing releases unless you have a backup somewhere!

Start by installing ownCloud, either on real hardware or in a VM. You can find instructions for installating ownCloud in the Manual Installation on Linux or Linux Package Manager Installation

Please note that we are still working on the documentation and if you bump into a problem, you can help us fix it. Small things can be edited straight on GitHub.

The Real Testing

Testing is a matter of trying out some scenarios you decide on or were asked to test, for example, sharing a folder and mounting it on another ownCloud instance. If it works – awesome, move on. If it doesn't, find out as much as you can about why it doesn't and use that for a bug report.

This is the stage where you should see if your issue is already reported by checking the relevant bug tracker. It might even be fixed, sometimes! Alternatively, just ask on the test-pilots mailing list.

Finally, if the issue you bump into is a definite bug and the developers are not aware of it, file it as a new issue in the relevant bug tracker.

Unit-Testing

PHP Unit Tests

ownCloud uses PHPUnit >= 4.8 for unit testing PHP code.

Getting PHPUnit

ownCloud >= 10.0

If you are using ownCloud 10.0 or higher, running make in your terminal from the webroot directory will prepare everything for testing. This will install beside necessary dependencies, a local version of PHPUnit at <webroot>/lib/composer/phpunit/phpunit.

- Run make help to get a list of parameters
- To update your testing environment run make clean and make again.
- Take care that the php phpunit file in the path provided has the executable permission set.

ownCloud < 10.0

If you are on any version earlier than 10.0 you have to setup PHPUnit (and run the tests) manually. There are three ways to install it:

1. Use Composer

```
composer require phpunit/phpunit
```

1. Use your package manager (if you're using a Linux distribution)

```
# When using a Debian-based distribution
sudo apt-get install phpunit
```

1. Install it manually

```
wget https://phar.phpunit.de/phpunit.phar
chmod +x phpunit.phar
sudo mv phpunit.phar /usr/local/bin/phpunit
```

After the installation the command phpunit is available

```
phpunit --version
```

Please be aware that PHPUnit 6.0 and above require PHP 7.0.

And you can update it using:

```
phpunit --self-update
```

This option is not supported from PHPUnit 6.0 onward. If you're using this version or higher, please use either Composer or your package manager to upgrade to the latest version.

You can find more information in the PHPUnit documentation.

Running PHP Unit tests for ownCloud >= 10.0

There are existing tests provided by ownCloud which are ready to run.

- Change into webroot and run make help to see tests and parameters available.

Testing apps

- To run the tests for a specific app with the provided PHPUnit version, change into <webroot>/apps/<appname> and

```
make test-php
```

Writing PHP Unit tests

To get started, do the following:

- Create a directory called tests/unit in the top level of your application
- Create a PHP file in the directory and require_once your class which you want to test.

Then you can run the created test with phpunit.

If you use ownCloud functions in your class under test (i.e: OC::getUser()) you'll need to bootstrap ownCloud or use dependency injection.

You'll most likely run your tests under a different user than the Web server. This might cause problems with your PHP settings (i.e., open_basedir) and requires you to adjust your configuration.

Given the class MyClass in your app:

Listing 1. /srv/http/owncloud/apps/myapp/tests/lib/MyClass.php

```
Unresolved directive in modules/developer_manual/pages/testing/unit-testing.adoc - include::/drone/src/modules/developer_manual/examples/core/unit-testing/MyClass.php[MyClass.php]
```

An example for a simple test would be:

Listing 2. /srv/http/owncloud/apps/myapp/tests/unit/MyClassTest.php

```
Unresolved directive in modules/developer_manual/pages/testing/unit-testing.adoc - include::/drone/src/modules/developer_manual/examples/core/unit-testing/MyClassTest.php[MyClassTest.php]
```

The class under test and the test class should share the same namespace so you do not need to use a dedicated use statement for it. This is the recommended way to organize tests.

In /srv/http/owncloud/apps/myapp/ you run the test with the following command:

```
phpunit tests/unit/MyClassTest.php
```

Make sure to extend the \Test\TestCase class with your test and always call the parent methods, when overwriting setUp(), setUpBeforeClass(), tearDown() or tearDownAfterClass() method from the TestCase. These methods set up important stuff and clean up the system after the test so that the next test can run without side effects, such as clearing files and entries from the file cache, etc. For more resources on writing tests for PHPUnit visit the writing tests section of the PHPUnit documentation.

Bootstrapping ownCloud

If you use ownCloud functions or classes in your code, you'll need to make them available to your test by bootstrapping ownCloud.

To do this, you'll need to provide the --bootstrap argument when running PHPUnit

/srv/http/owncloud

```
phpunit --bootstrap tests/bootstrap.php apps/myapp/tests/testsuite.php
```

If you run the test suite as a user other than your Web server, you'll have to adjust your php.ini and file rights.

/etc/php/php.ini

```
open_basedir = none
```

/srv/http/owncloud:

```
su -c "chmod a+r config/config.php"
su -c "chmod a+rx data/"
su -c "chmod a+w data/owncloud.log"
```

Running Unit Tests for ownCloud Core

The core project provides a script that runs all the core unit tests using the specified database backend like sqlite, mysql, pgsql, oci (for Oracle), the default is sqlite

To run tests on mysql or pgsql you need a database user called oc_autotest with the password owncloud. This user needs the privilege to create and delete the database called oc_autotest.

MySQL Setup

```
CREATE DATABASE oc_autotest;
CREATE USER 'oc_autotest'@'localhost' IDENTIFIED BY 'owncloud';
GRANT ALL ON oc_autotest.* TO 'oc_autotest'@'localhost';
```

For parallel executor support with EXECUTOR_NUMBER=0

```
CREATE DATABASE oc_autotest0;
CREATE USER 'oc_autotest0'@'localhost' IDENTIFIED BY 'owncloud';
GRANT ALL ON oc_autotest0.* TO 'oc_autotest0'@'localhost';
```

PostgreSQL Setup

```
su - postgres

# Use password "owncloud"
createuser -P oc_autotest

# Give the user the privilege to create databases
psql -c 'ALTER USER oc_autotest CREATEDB;'
```

To enable dropdb add local all all trust to pg_hba.conf.

For parallel executor support with EXECUTOR_NUMBER=0

```
su - postgres

# Use password "owncloud"
createuser -P oc_autotest0

# Give the user the privilege to create databases
psql -c 'ALTER USER oc_autotest0 CREATEDB;'
```

Run Tests

To run all tests, run the following command:

```
make test-php
```

To run tests only for MySQL, run the following command:

```
make test-php TEST_DATABASE=mysql
```

To run a particular test suite, use the following command as a guide:

```
make test-php TEST_DATABASE=mysql TEST_PHP_SUITE=tests/lib/share/share.php
```

By default, a code coverage report is generated after the test run. To avoid the time taken for that, specify NOCOVERAGE:

```
make test-php NOCOVERAGE=true TEST_DATABASE=mysql
TEST_PHP_SUITE=tests/lib/share/share.php
```

Further Reading

- Writing Testable Code
- PHPUnit Manual
- Clean Code Talks - GuiceBerry
- Clean Code by Robert C. Martin

Unit Testing JavaScript in Core

JavaScript Unit testing for **core** and **core apps** is done using the Karma test runner with Jasmine.

Installing Node JS

To run the JavaScript unit tests you will need to install **Node JS**. You can get it here: http://nodejs.org/ After that you will need to setup the **Karma** test environment. The easiest way to do this is to run the automatic test script first, see next section.

Running All The Tests

To run all JavaScript tests, run the following command:

```
make test-js
```

This will also automatically set up your test environment.

Debugging Tests in the Browser

To debug tests in the browser, this will run **Karma** in browser mode

```
make test-js-debug
```

From there, open the URL http://localhost:9876 in a web browser. On that page, click on the Debug button. An empty page will appear, from which you must open the browser console (F12 in Firefox/Chrome). Every time you reload the page, the unit tests will be relaunched and will output the results in the browser console.

Unit Test File Paths

JavaScript unit test examples can be found in apps/files/tests/js/ Unit tests for the core app JavaScript code can be found in core/js/tests/specs

Documentation

Here are some useful links about how to write unit tests with Jasmine and Sinon:

- Karma test runner: http://karma-runner.github.io
- Jasmine: https://jasmine.github.io
- Sinon (for mocking and stubbing): http://sinonjs.org/

Acceptance Tests

The Test Directory Structure

This is the structure of the acceptance directory inside the core repository's tests directory:

```
tests
├── acceptance
│   ├── config
│   │   └── behat.yml
│   ├── features
│   │   ├── apiTags (example suite of API tests)
│   │   │   └── feature files (behat gherkin files)
│   │   ├── bootstrap
│   │   │   └── Contexts and traits (php files)
│   │   ├── cliProvisioning (example suite of CLI tests)
│   │   │   └── feature files (behat gherkin files)
│   │   ├── lib
│   │   │   └── Page objects for webUI tests (php files)
│   │   └── webUILogin (example suite of webUI tests)
│   │       └── feature files (behat gherkin files)
│   ├── filesForUpload
│   └── run.sh
```

Here's a short description of each component of the directory.

config/

This directory contains behat.yml which sets up the acceptance tests. In this file we can add new suites and define the contexts needed by each suite. Here's an example configuration:

```
default:
  autoload:
    '': '%paths.base%/../features/bootstrap'
  suites:
    apiMain:
      paths:
        - '%paths.base%/../features/apiMain'
      contexts:
        - FeatureContext: &common_feature_context_params
            baseUrl:  http://localhost:{std-port-http}
            adminUsername: admin
            adminPassword: admin
            regularUserPassword: 123456
            ocPath: apps/testing/api/v1/occ
        - AppManagementContext:
        - CalDavContext:
        - CardDavContext:

    apiCapabilities:
      paths:
        - '%paths.base%/../features/apiCapabilities'
      contexts:
        - FeatureContext: *common_feature_context_params
        - CapabilitiesContext:
```

features/

This directory contains sub-directories for each of the test suites.

features/suiteName

This directory stores Behat's feature files for the test suite. These contain Behat's test cases, called scenarios, which use the Gherkin language.

feature/bootstrap

This folder contains all the Behat contexts. Contexts contain the PHP code required to run Behat's scenarios. Every suite has to have one or more contexts associated with it. The contexts define the test steps used by the scenarios in the feature files of the test suite.

filesForUpload/

This folder contains convenience files that tests can use to upload.

run.sh

This script runs the test suites. It is called by the make commands that are used to run acceptance tests.

The Testing App

The testing app provides an API that allows the acceptance tests to set up the environment of the system-under-test. For example, running occ commands to set

system and app config settings. The testing app must be installed and enabled on the system-under-test.

The testing app also provides skeleton folders that the tests can use as the default set of files for new users.

apps/testing/data/apiSkeleton/

This folder stores the initial files loaded for a new user during API or CLI acceptance tests.

apps/testing/data/webUISkeleton/

This folder stores the initial files loaded for a new user during webUI acceptance tests.

Running Acceptance Tests

Preparing to Run Acceptance Tests

This is a concise guide to running acceptance tests on ownCloud 10. Before you can do so, you need to meet a few prerequisites available; these are

- ownCloud
- Composer
- MySQL

In php.ini on your system, set opcache.revalidate_freq=0 so that changes made to ownCloud config.php by test scenarios are implemented immediately.

After cloning core, run make as your webserver's user in the root directory of the project.

Now that the prerequisites are satisfied, and assuming that $installation_path is the location where you cloned the ownCloud/core repository, the following commands will prepare the installation for running the acceptance tests.

```
# Remove current configuration (if existing)
sudo rm -rf $installation_path/data/*
sudo rm -rf $installation_path/config/*

# Remove existing 'owncloud' database
mysql -u root -h localhost -e "drop database owncloud"
mysql -u root -h localhost -e "drop user oc_admin"
mysql -u root -h localhost -e "drop user oc_admin@localhost"

# Install ownCloud server with the command-line
sudo -u www-data $installation_path/occ maintenance:install \
  --database='mysql' --database-name='owncloud' --database-user='root' \
  --database-pass='mysqlrootpassword' --admin-user='admin' --admin-pass
='admin'
```

Types of Acceptance Tests

There are 3 types of acceptance tests; API, CLI and webUI.

- API tests test the ownCloud public APIs.

- CLI tests test the occ command-line commands.
- webUI tests test the browser-based user interface.

webUI tests require an additional environment to be set up. See the UI testing documentation for more information. API and CLI tests are run by using the test-acceptance-api and test-acceptance-cli make commands.

Running Acceptance Tests for a Suite

Run a command like the following:

```
make test-acceptance-api BEHAT_SUITE=apiTags
make test-acceptance-cli BEHAT_SUITE=cliProvisioning
```

Running Acceptance Tests for a Feature

Run a command like the following:

```
make test-acceptance-api BEHAT_FEATURE
=tests/acceptance/features/apiTags/createTags.feature
make test-acceptance-cli BEHAT_FEATURE
=tests/acceptance/features/cliProvisioning/addUser.feature
```

Running Acceptance Tests for a Tag

Some test scenarios are tagged. For example, tests that are known to fail and are awaiting fixes are tagged @skip. To run test scenarios with a particular tag:

```
make test-acceptance-api BEHAT_SUITE=apiTags BEHAT_FILTER_TAGS=@skip
make test-acceptance-cli BEHAT_SUITE=cliProvisioning BEHAT_FILTER_TAGS=@skip
```

Displaying the ownCloud Log

It can be useful to see the tail of the ownCloud log when the test run ends. To do that, specify SHOW_OC_LOGS=true:

```
make test-acceptance-api BEHAT_SUITE=apiTags SHOW_OC_LOGS=true
```

Optional Environment Variables

If you want to use an alternative home name using the env variable add to the execution OC_TEST_ALT_HOME=1, as in the following example:

```
make test-acceptance-api BEHAT_SUITE=apiTags OC_TEST_ALT_HOME=1
```

If you want to have encryption enabled add OC_TEST_ENCRYPTION_ENABLED=1, as in the following example:

```
make test-acceptance-api BEHAT_SUITE=apiTags OC_TEST_ENCRYPTION_ENABLED
=1
```

How to Write Acceptance Tests

Each acceptance test is a scenario in a feature file in a test suite.

Feature Files

Each feature file describes and tests a particular feature of the software. The feature file starts with the Feature: keyword, a sentence describing the feature. This is followed by more detail explaining who uses the feature and why, in the format:

```
  As a [role]
  I want [feature]
  So that [benefit]
```

For example:

```
Feature: upload file using the WebDav API
  As a user
  I want to be able to upload files
  So that I can store and share files between multiple client systems
```

This detail is free-text and has no effect on the running of automated tests.

The rest of a feature file contains the test scenarios.

Make small feature files for individual features. For example "the Provisioning API" is too big to be a single feature. Split it into the functional things that it allows a client to do. For example:

- addGroup.feature
- addUser.feature
- addToGroup.feature
- deleteGroup.feature
- deleteUser.feature
- disableUser.feature
- editUser.feature
- enableUser.feature
- removeFromGroup.feature

Test Scenarios

A feature file should have up to 10 or 20 scenarios that test the feature. If you need more scenarios than that, then perhaps there really are multiple features and you should make multiple feature files.

Each scenario starts with the Scenario: keyword followed by a description of the scenario. Then the steps to execute for that scenario are listed.

There are 3 types of test steps:

- Given steps that get the system into the desired state to start the test (e.g. create users and groups, share some files)
- When steps that perform the action under test (e.g. upload a file to a share)
- Then steps that verify that the action was successful (e.g. check the HTTTP status code, check that other users can access the uploaded file)

A single scenario should test a single action or logical sequence of actions. So the Given, When and Then steps should come in that order.

If there are multiple Given or When steps, then steps after the first start with the keyword And.

If there are multiple Then steps, then steps after the first start with the keyword And or But.

Writing a Given Step

Given steps are written in the present-perfect tense. They specify things that "have been done". For example:

```
  Scenario: delete files in a sub-folder
    Given user "user0" has been created
    And user "user0" has moved file "/welcome.txt" to "/FOLDER/welcome.txt"
    And user "user0" has created a folder "/FOLDER/SUBFOLDER"
    And user "user0" has copied file "/textfile0.txt" to
"/FOLDER/SUBFOLDER/testfile0.txt"
```

Given steps do not mention how the action is done. They can mention the actor that performs the step, when that matters. For example, creating a user must be done by something with enough admin privilege. So there is no need to mention "the administrator". But creating a file must be done in the context of some user. So the user must be mentioned.

The test code is free to achieve the desired system state however it likes. For example, by using an available API, by running a suitable occ command on the system-under-test, or by doing it with the webUI. Typically the test code for Given steps will use an API, because that is usually the most efficient.

Writing a When Step

When steps are written in the simple present tense. They specify the action that is being tested. Continuing the example above:

```
  Scenario: delete all files in a sub-folder
    Given user "user0" has been created
    And user "user0" has moved file "/welcome.txt" to "/FOLDER/welcome.txt"
    And user "user0" has created a folder "/FOLDER/SUBFOLDER"
    And user "user0" has copied file "/textfile0.txt" to
"/FOLDER/SUBFOLDER/testfile0.txt"
    When user "user0" deletes everything from folder "/FOLDER/" using the WebDAV
API
```

In ownCloud there are usually 2 or 3 interfaces that can implement an action. For example, a user can be created using an occ command, the Provisioning API or the webUI. Files can be managed using the WebDAV API or the webUI. File shares can be managed using the Sharing API or the webUI. So When steps should end with a phrase specifying the interface to be tested, such as:

- using the occ command
- using the Sharing API
- using the Provisioning API
- using the WebDAV API
- using the webUI

Writing a Then Step

Then steps describe what should be the case if the When step(s) happened successfully. They should contain the word should somewhere in the step text.

```
  Scenario: delete all files in a sub-folder
    Given user "user0" has been created
    And user "user0" has moved file "/welcome.txt" to "/FOLDER/welcome.txt"
    And user "user0" has created a folder "/FOLDER/SUBFOLDER"
    And user "user0" has copied file "/textfile0.txt" to
"/FOLDER/SUBFOLDER/testfile0.txt"
    When user "user0" deletes everything from folder "/FOLDER/" using the WebDAV
API
    Then user "user0" should see the following elements
      | /FOLDER/          |
      | /PARENT/          |
      | /PARENT/parent.txt |
      | /textfile0.txt    |
      | /textfile1.txt    |
      | /textfile2.txt    |
      | /textfile3.txt    |
      | /textfile4.txt    |
    But user "user0" should not see the following elements
      | /FOLDER/SUBFOLDER/             |
      | /FOLDER/welcome.txt            |
      | /FOLDER/SUBFOLDER/testfile0.txt |
```

Note that there are often multiple things that should or should not be the case after the When action. For example, in the above scenario, various files and folders (that are part of the skeleton) should still be there. But other files and folders under FOLDER should have been deleted.

Where it makes the scenario read more easily, use the But as well as And keywords in the Then section.

Then steps should test an appropriate range of evidence that the When action did happen. For example:

```
  Scenario: admin creates a user
    Given user "brand-new-user" has been deleted
    When the administrator sends a user creation request for user "brand-new-user"
password "%alt1%" using the provisioning API
    Then the OCS status code should be "100"
    And the HTTP status code should be "200"
    And user "brand-new-user" should exist
    And user "brand-new-user" should be able to access a skeleton file
```

In this scenario we check that the OCS and HTTP status codes of the API request are good. But it is possible that the server lies, and returns HTTP status 200 for every request, even if the server did not create the user. So we check that the user exists. However maybe the user exists according to some API that can query for valid user names/ids, but the user account is not really valid and working. So we also check that the user can do something, in this case that they can access one of their skeleton files.

Specifying the Actor

Test steps often need to specify the actor that does the action or check. For example, the user.

The acceptance test code can remember the "current" user with a step like:

```
    Given as user "user0"
    And the user has uploaded file "abc.txt"
    When the user deletes file "abc.txt"
    ...
```

So that later steps can just mention the user.

Or you can mention the user in each step:

```
    Given user "user0" has uploaded file "abc.txt"
    When user "user0" deletes file "abc.txt"
    ...
```

Either form is acceptable. Longer tests with a single user read well with the first form. Shorter tests, or sharing tests that mix actions of multiple users, read well with the second form.

When the actor is the administrator (a special user with privileges) then use the administrator in the step text. Do not write When user "admin" does something. The user name of the user with administrator privilege on the system-under-test might not be admin. The user name of the administrator needs to be determined at run-time, not hard-coded in the scenario.

Referring to Named Entities

When referring to specific named entities on the system, such as a user, group, file, folder or tag, then do not put the word the in front, but do put the name of the entity. For example:

```
Given user "user0" has been added to group "grp1"
And user "user0" has uploaded file "abc.txt" into folder "folder1"
And user "user0" has added tag "aTag" to file "folder1/abc.txt"
When user "user0" shares folder "folder1" with user "user1"
...
```

This makes it clearer to understand which entity is required in which position of the sentence. For example:

```
And "user0" has uploaded "abc.txt" into "folder1"
...
```

would be less clear that the required entities for this step are a user, file and folder.

Scenario Background

If all the scenarios in a feature start with a common set of Given steps, then put them into a Background: section. For example:

```
Background:
  Given user "user0" has been created
  And user "user1" has been created
  And user "user0" has uploaded file "abc.txt"

Scenario: share a file with another user
  When user "user0" shares file "abc.txt" with user "user1" using the sharing API
  Then the HTTP status code should be "200"
  And user "user1" should be able to download file "abc.txt"

Scenario: share a file with a group
  Given group "grp1" has been created
  And "user1" has been added to group "grp1"
  When user "user0" shares file "abc.txt" with user "user1" using the sharing API
  Then the HTTP status code should be "200"
  And user "user1" should be able to download file "abc.txt"
```

This reduces some duplication in feature files.

Controlling Running Test Scenarios In Different Environments

A feature or test scenario might only be relevant to run on a system-under-test that has a particular environment. For example, a particular app enabled.

To allow the test runner script to run the features and scenarios relevant to the system-under-test the feature file or individual scenarios are tagged. The test runner script can then filter by tags to select the relevant features or scenarios.

For general information on tagging features and scenarios see the Behat tags documentation.

Tagging Features By API, CLI and webUI

Tag every feature with its major acceptance test type api, cli or webUI, as in the following examples. Doing so allows the tests of a particular major type to be quickly run or skipped.

@api

```
@api
Feature: add groups
  As an admin
  I want to be able to add groups
  So that I can more easily manage access to resources by groups rather than individual users
```

@cli

```
@cli
Feature: add group
  As an admin
  I want to be able to add groups
  So that I can more easily manage access to resources by groups rather than individual users
```

@webUI

```
@webUI
Feature: login users
  As a user
  I want to be able to log into my account
  So that I have access to my files
```

Tagging Scenarios That Require An App

When a feature or scenario requires a core app to be enabled then tag it like:

```
@comments-app-required
@federation-app-required
@files_trashbin-app-required
@files_versions-app-required
@notifications-app-required
@provisioning-app-required
@systemtags-app-required
```

The above apps might be disabled on a system-under-test. Tagging the feature or scenario allows all tests for the app to be quickly run or skipped.

For tests in an app repository, do not tag them with the app name (e.g., files_texteditor-app-required). It is already a given that the app in the repository is required for running the tests!

Tagging Scenarios That Need to Be Skipped

Skip UI Tests On A Particular Browser

Some browsers have difficulty with some automated test actions. To skip scenarios for a browser tag them with the relevant tags:

```
@skipOnCHROME
@skipOnFIREFOX
@skipOnINTERNETEXPLORER
@skipOnMICROSOFTEDGE
```

Skip Tests On A Particular Version Of ownCloud

The acceptance test suite is sometimes run against a system-under-test that has an older version of ownCloud. When writing new test scenarios for a new or changed feature, tag them to be skipped on the previous recent release of ownCloud. Use tag formats like the following to skip on a particular major, minor or patch version.

```
@skipOnOcV10
@skipOnOcV10.0
@skipOnOcV10.0.10
@skipOnOcV10.1
```

Skip Tests In Other Environments

Annotation	Description
@skipOnLDAP	skip the scenario if the test is running with the LDAP backend. For example, some user provisioning features may not be relevant when LDAP is the backend for authentication.
@skipOnStorage:ceph	skip the scenario if the test is running with ceph backend storage.
@skipOnStorage:scality	skip the scenario if the test is running with scality backend storage.
@skipOnEncryption	skip the scenario if the test is running with encryption enabled.
@skipOnEncryptionType:masterkey	skip the scenario if the test is running with masterkey encryption enabled.
@skipOnEncryptionType:user-keys	skip the scenario if the test is running with user-keys encryption enabled.

Tags For Tests To Run In Special Environments

Annotation	Description
@smokeTest	this scenario has been selected as part of a base set of smoke tests.

Annotation	Description
@TestAlsoOnExternalUserBackend	this scenario is selected as part of a base set of tests to run when a special user backend is in place (e.g., LDAP).
@local_storage	this scenario requires and tests the local storage feature.

Special Tags for UI Tests

Annotation	Description
@insulated	this makes the browser driver restart the browser session between each scenario. It helps isolate the browser state. When the browser session is recording, there is a separate video for each scenario. Use this tag on all UI scenarios.
@disablePreviews	generating previews/thumbnails takes time. Use this tag on UI test scenarios that do not need to test thumbnail behavior.

Writing Scenarios For Bugs

If you are developing a new feature, and the scenarios that you have written do not pass, or existing scenarios are failing, then fix the code so that they pass.

If you are writing scenarios to cover features and scenarios that are not currently covered by acceptance tests then you may find existing bugs.

If the bug is easy to fix, then provide the bugfix and the new acceptance test scenario(s) in the same pull request.

If the bug is not easy to fix, then:

- create an issue describing the bug.
- write a scenario that demonstrates the existing wrong behavior.
- include commented-out steps in the scenario to document what is the expected correct behavior.
- write the scenario so that it will fail when the bug is fixed.
- tag the scenario with the issue number.

```
  @issue-32385
  Scenario: Change email address
    When the user changes the email address to "new-address@owncloud.com"
using the webUI
    # When the issue is fixed, remove the following step and replace with the
commented-out step
    Then the email address "new-address@owncloud.com" should not have received
an email
    #And the user follows the email change confirmation link received by "new-
address@owncloud.com" using the webUI
    Then the attributes of user "user1" returned by the API should include
      | email | new-address@owncloud.com |
```

The above scenario is an example of this. When the bug is fixed then the step about should not have received an email will fail. CI will fail, and so the developer will notice this scenario and will have to correct it.

See the Behat User Guide for information about writing test step code.

In addition to that, follow these guidelines.

Given Steps

The code of a Given step should achieve the desired system state by whatever means is quick to execute. Typically use a public API if available, rather than running an occ command via the testing app or entering data in the webUI.

If there is a simple way to gain confidence that the Given step was successful, then do it. Typically this will check a status code returned in the API response. Doing simple confidence checks in Given steps makes it easier to catch some unexpected problem during the scenario Given section.

Here's example code for a Given step:

```
/**
 * @Given the administrator has changed the password of user :user to :password
 *
 * @param string $user
 * @param string $password
 *
 * @return void
 * @throws \Exception
 */
public function adminHasChangedPasswordOfUserTo(
    $user, $password
) {
    $this->adminChangesPasswordOfUserToUsingTheProvisioningApi(
        $user, $password
    );
    $this->theHTTPStatusCodeShouldBe(
        200,
        "could not change password of user $user"
    );
}
```

The code calls the method for the When step and then checks the HTTP status code.

When Steps

The code of a When step should perform the action but not check its result. A When step should not ordinarily fail. Often a When step will save the response. It is the responsibility of later Then steps to decide if the scenario passed or failed.

Here's example code for a When step:

```
/**
 * @When the administrator changes the password of user :user to :password using
the provisioning API
 *
 * @param string $user
 * @param string $password
 *
 * @return void
 * @throws \Exception
 */
public function adminChangesPasswordOfUserToUsingTheProvisioningApi(
    $user, $password
) {
    $this->response = UserHelper::editUser(
        $this->getBaseUrl(),
        $user,
        'password',
        $password,
        $this->getAdminUsername(),
        $this->getAdminPassword()
    );
}
```

The code saves the response so that later Then steps can examine it.

Then Steps

The code of a Then step should check some result of the When action. Often it will find information in the saved response and assert something.

Here's example code for a Then step:

```
/**
 * @Then /^the groups returned by the API should include "([^"]*)"$/
 *
 * @param string $group
 *
 * @return void
 */
public function theGroupsReturnedByTheApiShouldInclude($group) {
    $respondedArray = $this->getArrayOfGroupsResponded($this->response);
    PHPUnit_Framework_Assert::assertContains($group, $respondedArray);
}
```

However, a Then step may need to do actions of its own to retrieve more information about the state of the system. For example, after changing a user password we could check that the user can still access some file:

```
/**
 * @Then /^as "([^"]*)" (file|folder|entry) "([^"]*)" should exist$/
 *
 * @param string $user
 * @param string $entry
 * @param string $path
 *
 * @return void
 * @throws \Exception
 */
public function asFileOrFolderShouldExist($user, $entry, $path) {
   $path = $this->substituteInLineCodes($path);
   $this->responseXmlObject = $this->listFolder($user, $path, 0);
   PHPUnit_Framework_Assert::assertTrue(
      $this->isEtagValid(),
      "$entry '$path' expected to exist but not found"
   );
}
```

In the above example, listFolder is called and does an API call to access the file and then asserts that the response has a valid ETag.

References

For more information on Behat, and how to write acceptance tests using it, see the Behat documentation. For background information on Behaviour-Driven Development (BDD), see Dan North resources.

User Interface Testing

Requirements

- ownCloud >= 10.0. Make sure you have a running instance of ownCloud setup completely.
- Default language set to en (in config/config.php set 'default_language' ⇒ 'en',).
- An admin user called admin with the password admin.
- No self-signed SSL certificates.
- The testing app installed and enabled.
- Testing utils (running make in your terminal from the webroot directory will install them).
- Docker CE Installed
- Docker Post-install done to put your developer account in the docker group so you can run Docker without sudo
- Docker subnet enabled for any firewall that may be active such as, ufw. The example below shows how to update ufw's firewall rules to allow the 172.17.0.0/16 Docker subnet:

```
sudo ufw status
sudo ufw allow from 172.17.0.0/16
```

- Docker containers pulled. It is recommended to use standalone-chrome-debug which allows seeing the browser live. The latest standalone-chrome-* containers have an issue. So make sure to pull the specific chrome container versions listed below. You will also need MailHog. Pull any or all of these Docker containers:

```
docker pull selenium/standalone-chrome:3.141.59-oxygen
docker pull selenium/standalone-chrome-debug:3.141.59-oxygen
docker pull selenium/standalone-firefox
docker pull selenium/standalone-firefox-debug
docker pull mailhog/mailhog
```

- A vnc viewer installed (in order to view the browser action as the UI tests run). For example:

```
sudo apt install tigervnc-viewer
```

- To run the Selenium server locally (not in Docker) see the notes at the end.

Overview

Tests are divided into suites, enabling each suite to test some logical portion of the functionality and for the total elapsed run-time of a single suite to be reasonable (up to about 40 minutes on Travis-CI, about 10 minutes on drone). Elapsed run-time on a local developer system is very dependent on the IO as well as CPU performance. Smaller apps may have all tests in a single suite.

Each suite consists of a number of features. Each feature is described in a *.feature file. There are a number of scenarios in each feature file. Each scenario has a number of scenario steps that define the steps taken to do the test.

Set Up Test

- Start the Selenium Docker container in a terminal:

```
docker run -p 4445:4444 -p 5900:5900 -v /dev/shm:/dev/shm selenium/standalone-chrome-debug
```

Ports on the Selenium Docker IP address are mapped to localhost so they can be accessed by the tests and the vnc viewer.

- Start the MailHog Docker container in another terminal:

```
docker run -p 1025:1025 -p 8025:8025 mailhog/mailhog
```

Ports on the MailHog docker IP address are mapped to localhost so they can be accessed by the tests. By running these in terminal windows, it is simple to press ctrl-C to stop them when you are finished.

- Set the following environment variables:
  - TEST_SERVER_URL (The URL of your webserver)
  - TEST_SERVER_FED_URL (The alternative URL of your webserver for federation share tests.)

- BROWSER (Any one of chrome, firefox, internet explorer or MicrosoftEdge. Defaults to chrome)
- BROWSER_VERSION (version of the browser you want to use - *optional*)

e.g., to test an instance running on the Docker subnet with Chrome do:

```
export TEST_SERVER_URL=http://172.17.0.1:{std-port-http}/owncloud-core
export TEST_SERVER_FED_URL=http://172.17.0.1:8180/owncloud-core
export BROWSER=chrome
```

- If your ownCloud install is running locally on Apache, then it should already be available on the Docker subnet at 172.17.0.1
- To run the federation Sharing tests:
  1. Make sure you have configured HTTPS with valid certificates on both servers URLs
  2. Import SSL certificates (or do not offer HTTPS).
- Run a suite of tests:

```
make test-acceptance-webui BEHAT_SUITE=webUILogin
```

The names of suites are found in the tests/acceptance/config/behat.yml file, and start with webUI.

- The browser for the tests runs inside the Selenium docker container. View it by running the vnc viewer: vncviewer.

And connect to localhost. The VNC password of the docker container is secret.

Running UI Tests using IPv6

The test system must have (at least locally) functioning IPv6:

- working loopback address ::1
- a real routable IPv6 address (not just a link-local address)

If you have a server set up that listens on both IPv4 and IPv6 (e.g. localhost on 127.0.0.1 and ::1) then the UI tests will access the server via whichever protocol your operating system prefers. If there are tests that specifically specify IPv4 or IPv6, then those will choose a suitable local address to come from so that they access the server using the required IP version.

If you are using the PHP dev server, then before starting it, in addition to the exports in the Set Up Test section, specify where the IPv6 server should listen:

```
export IPV6_HOST_NAME=ip6-localhost
```

Then both IPv4 and IPv6 PHP dev servers will be started by the script:

```
bash tests/travis/start_php_dev_server.sh
```

If you want the tests to drive the UI over IPv6, then export an IPv6 name or address

for SRV_HOST_NAME and an IPv4 name or address for IPV4_HOST_NAME:

```
export SRV_HOST_NAME=ip6-localhost
export IPV4_HOST_NAME=localhost
```

Because not everyone will have functional IPv6 on their test system yet, tests that specifically require IPv6 are tagged @skip @ipv6. To run those tests, follow the section below on running skipped tests and specify --tags @ipv6.

Running UI Tests for One Feature

You can run the UI tests for just a single feature by specifying the feature file:

```
make test-acceptance-webui
BEHAT_FEATURE=tests/acceptance/features/webUITrashbin/trashbinDelete.feature
```

To run just a single scenario within a feature, specify the line number of the scenario:

```
make test-acceptance-webui
BEHAT_FEATURE=tests/acceptance/features/webUITrashbin/trashbinDelete.feature
<linenumber>
```

Running UI Tests for an App

With the app installed, run the UI tests for the app from the app root folder:

```
cd apps/files_texteditor
../../tests/acceptance/run.sh --suite webUITextEditor
```

Run UI the tests for just a single feature of the app by specifying the feature file:

```
cd apps/files_texteditor
../../tests/acceptance/run.sh
tests/acceptance/features/webUITextEditor/editTextFiles.feature
```

Skipping Tests

If a UI test is known to fail because of an existing bug, then it is left in the test set *but* is skipped by default. Skip a test by tagging it @skip and then put another tag with text that describes the reason it is skipped. e.g.,:

```
@skip @trashbin-restore-problem-issue-1234
Scenario: restore a single file from the trashbin
```

Skipped tests are listed at the end of a default UI test run. You can locally run the skipped test(s). Run all skipped tests for a suite with:

```
make test-acceptance-webui BEHAT_SUITE=webUITrashbin
BEHAT_FILTER_TAGS=@skip
```

Or run just a particular test by using its unique tag:

```
make test-acceptance-webui BEHAT_SUITE=webUITrashbin
BEHAT_FILTER_TAGS=@trashbin-restore-problem-issue-1234
```

When fixing the bug, remove these skip tags in the PR along with the bug fix code.

Additional Command Options

Running all test suites in a single run is not recommended. It will take more than 1 hour on a typical development system. However, you may run all UI tests with:

```
make test-acceptance-webui
```

By default, any test scenarios that fail are automatically rerun once. This minimizes transient failures caused by browser and Selenium driver timing issues. When developing tests it can be convenient to override this behavior.

To not rerun failed test scenarios:

```
make test-acceptance-webui NORERUN=true BEHAT_SUITE=webUILogin
```

Local Selenium Setup

You may optionally run the Selenium server locally. Docker is now the recommended way, but local Selenium is also possible:

- Selenium standalone server e.g. version 3.12.0 or newer.
- Browser installed that you would like to test on (e.g. chrome)
- Web driver for the browser that you want to test.
- Place the Selenium standalone server jar file and the web driver(s) somewhere in the same folder.
- Start the Selenium server:

```
java -jar selenium-server-standalone-3.12.0.jar \
  -port 4445 \
  -enablePassThrough false
```

- In this configuration, the tests will continually open the browser-under-test on your local system.
- If you run any test scenarios that need MailHog (to test password reset etc.), then you need to run the MailHog Docker container. That is much simpler than trying to configure MailHog on your local system.

Known Issues

- Tests that are known not to work in specific browsers are tagged e.g., @skipOnFIREFOX47+ or @skipOnINTERNETEXPLORER and will be skipped by the script automatically
- - The web driver for the current version of Firefox works differently to the old one. If you want to test FF < 56 you need to test on 47.0.2 and to use Selenium server 2.53.1 for it
  - Download and install version 47.0.2 of Firefox.
  - Download version 2.53.2 of the Selenium web driver.

Unresolved directive in modules/developer_manual/pages/core/apis/index.adoc - include::admin_manual:/drone/src/modules/developer_manual/pages/_partials/section_page.adoc[]

External API

Introduction

The external API inside ownCloud allows third party developers to access data provided by ownCloud apps. ownCloud follows the OCS v1.7 specification (draft).

Usage

Registering Methods

Methods are registered inside the appinfo/routes.php using :phpOCP\\API

```
<?php

\OCP\API::register(
   'get',
   '/apps/yourapp/url',
   function($urlParameters) {
     return new \OC_OCS_Result($data);
   },
   'yourapp',
   \OC_API::ADMIN_AUTH
);
```

Returning Data

Once the API backend has matched your URL, your callable function as defined in **$action** will be executed. This method is passed as array of parameters that you defined in **$url**. To return data back the the client, you should return an instance of :phpOC_OCS_Result. The API backend will then use this to construct the XML or JSON response.

Authentication & Basics

Because REST is stateless you have to send user and password each time you access the API. Therefore running ownCloud **with SSL is highly recommended** otherwise **everyone in your network can log your credentials**:

```
https://user:password@yourowncloud.com/ocs/v1.php/apps/yourapp
```

Output

The output defaults to XML. If you want to get JSON append this to the URL:

```
?format=json
```

Output from the application is wrapped inside a **data** element:

XML

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data>
   <!-- data here -->
 </data>
</ocs>
```

JSON

```
{
 "ocs": {
  "meta": {
   "status": "ok",
   "statuscode": 100,
   "message": null
  },
  "data": {
   // data here
  }
 }
}
```

Status codes

The status code can be any of the following numbers:

- **100** - successful
- **996** - server error
- **997** - not authorized
- **998** - not found

- **999** - unknown error

OCS Rest API

Available Capabilities

Request Path	Method	Content Type
/ocs/v1.php/cloud/capabilities?format=json	GET	text/plain

To retrieve a list of your ownCloud server's available capabilities, you need to make an authenticated GET request, as in the example below.

```
curl --silent -u admin:admin \
  'http://localhost/ocs/v1.php/cloud/capabilities?format=json' | json_pp
```

The example uses json_pp to make the response easier to read, and omits some content for the sake of brevity.

This will return a JSON response, similar to the example below, along with a status of: HTTP/1.1 200 OK.


```
{
  "ocs" : {
    "data" : {
      "capabilities" : {
        "notifications" : {
          "ocs-endpoints" : [
            "list",
            "get",
            "delete"
          ]
        },
        "files" : {
          "blacklisted_files" : [
            ".htaccess"
          ],
          "bigfilechunking" : true,
          "privateLinks" : true,
          "undelete" : true,
          "versioning" : true
        },
        "checksums" : {
          "preferredUploadType" : "SHA1",
          "supportedTypes" : [
            "SHA1"
          ]
        },
        "files_sharing" : {
          "default_permissions" : 31,
          "user" : {
```

```
      "send_mail" : false
    },
    "federation" : {
      "incoming" : true,
      "outgoing" : true
    },
    "resharing" : true,
    "user_enumeration" : {
      "enabled" : true,
      "group_members_only" : false
    },
    "api_enabled" : true,
    "group_sharing" : true,
    "share_with_group_members_only" : true,
    "public" : {
      "enabled" : true,
      "password" : {
        "enforced" : {
          "read_only" : true,
          "read_write" : true,
          "upload_only" : true
        },
        "enforced" : true
      },
      "multiple" : true,
      "social_share" : true,
      "send_mail" : false,
      "upload" : true,
      "expire_date" : {
        "enabled" : false
      },
      "supports_upload_only" : true
    }
  },
  "dav" : {
    "chunking" : "1.0"
  },
  "core" : {
    "webdav-root" : "remote.php/webdav",
    "status" : {
      "edition" : "Community",
      "installed" : "true",
      "needsDbUpgrade" : "false",
      "versionstring" : "10.0.3",
      "productname" : "ownCloud",
      "maintenance" : "false",
      "version" : "10.0.3.3"
    },
    "pollinterval" : 60
  }
```

```
      }
    }
  }
}
```

In the example, in the capabilities element, you can see that the server lists six capabilities, along with their settings, sub-settings, and their values.

Core

Stored under the core capabilities element, this returns the server's core status settings, the interval to poll for server side changes, and it's WebDAV API root.

Checksums

Stored under the checksums capabilities element, this returns the server's supported checksum types, and preferred upload checksum type.

Files

Stored under the files capabilities element, this returns the server's support for the following capabilities:

Capability	Response Key
Big file chunking	bigfilechunking
File versioning	versioning
Its ability to undelete files; and	undelete
The list of files that are currently blacklisted.	blacklisted_files

Files Sharing

Stored under the files_sharing capabilities element, this returns the server's support for file sharing, re-sharing (by users and groups), federated file support, and public link shares (as well as whether passwords and expiry dates are enforced), and also whether the sharing API is enabled.

Notifications

Stored under the notifications capabilities element, this returns what the server sends notifications for.

WebDAV

Stored under the dav capabilities element, this returns the server's WebDAV API support.

Other apps add detail information to the capabilities, to indicate the availability of certain features, for example notifications.

OCS Recipient API

Introduction

The OCS Recipient API is a new OCS endpoint that is used by the share dialog autocomplete process, when you pick a user or group to share to.

The base URL for all calls to the share API is:
<owncloud_base_url>/ocs/v1.php/apps/files_sharing/api/v1/sharees?format=json

Get Shares Recipients

Get All Shares

Get all shares from the user.

- Syntax: /shares
- Method: GET

Query Attributes

Attribute	Type	Description	Required	Default
format	string	The response format. Can be either xml or json		xml
search	string	The search string		
itemType	string	The type which is shared.	Yes	
		Can be either file or folder		
shareType	integer	Any one of:		
		- 0 (user)		
		- 1 (group)		
		- 6 (remote)		
page	integer	The page number in the results to be returned		1
perPage	integer	The number of items per page	Yes	200

Status Codes

Code	Description
100	Successful
400	Failure due to invalid query parameters

OCS Share API

Introduction

The OCS Share API allows you to access the sharing API from outside over pre-defined OCS calls. The base URL for all calls to the share API is:
/ocs/v1.php/apps/files_sharing/api/v1/shares/pending.

Get All Shares

Get all shares shared with a user.

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending
Method	GET

Request Attributes

Attribute	Type	Description
format	string	sets the output format of the response. Default value is xml. Available options are xml and json.
path	string	limit the shares to those in a specific path.
reshares	boolean	returns not only the shares shared with the current user but all shares.
shared_with_me	string	limits the returned shares to only those shared with the authenticating user.
state	string	limits the returned shares to only those with the specified state. Available options are accepted, all, declined, pending, and rejected. This attribute is only valid when shared_with_me is set. declined and rejected are interchangeable.
subfiles	boolean	returns all shares within a folder, given that path defines a folder. This option requires the path option to be specified.

Status Codes

Code	Description
100	Successful.
400	Not a directory (if the `subfile' argument was used).
404	Couldn’t fetch shares or file doesn’t exist.
997	Unauthorised.

Example Request Response Payloads

If the user that you’re connecting with is not authorized, then you will see output similar to the following:

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>997</statuscode>
  <message>Unauthorised</message>
 </meta>
 <data/>
</ocs>
```

If the user that you're connecting with *is* authorized, then you will see output similar to the following:

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data/>
</ocs>
```

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>404</statuscode>
  <message>wrong path, file/folder doesn't exist</message>
 </meta>
 <data/>
</ocs>
```

Listing 3. Files shared with the current user in XML format.

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data>
  <element>
   <id>115468</id>
   <share_type>3</share_type>
   <uid_owner>auser</uid_owner>
   <displayname_owner>A User</displayname_owner>
   <permissions>1</permissions>
   <stime>1481537775</stime>
   <parent/>
   <expiration/>
   <token>MMqyHrR0GTepo4B</token>
   <uid_file_owner>auser</uid_file_owner>
   <displayname_file_owner>A User</displayname_file_owner>
   <path>/Photos/Paris.jpg</path>
   <item_type>file</item_type>
   <mimetype>image/jpeg</mimetype>
   <storage_id>home::auser</storage_id>
   <storage>993</storage>
   <item_source>3994486</item_source>
   <file_source>3994486</file_source>
   <file_parent>3994485</file_parent>
   <file_target>/Shared/Paris.jpg</file_target>
   <share_with/>
   <share_with_displayname/>

<url>https://your.owncloud.install.com/owncloud/index.php/s/MMqyHrR0GTepo4B<
/url>
    <mail_send>0</mail_send>
  </element>
 </data>
</ocs>
```

Listing 4. Files shared with the current user in JSON format.


```
{
 "ocs": {
  "meta": {
   "status": "ok",
   "statuscode": 100,
   "message": null,
   "totalitems": "",
   "itemsperpage": ""
  },
  "data": [
   {
    "id": "1",
    "share_type": 0,
    "uid_owner": "testuser",
    "displayname_owner": "test user",
    "permissions": 19,
    "stime": 1564484858,
    "parent": null,
    "expiration": null,
    "token": null,
    "uid_file_owner": "testuser",
    "displayname_file_owner": "test user",
    "state": 1,
    "path": "/ownCloud Manual.pdf",
    "item_type": "file",
    "mimetype": "application/pdf",
    "storage_id": "home::testuser",
    "storage": 3,
    "item_source": 97,
    "file_source": 97,
    "file_parent": 57,
    "file_target": "/ownCloud Manual.pdf",
    "share_with": "admin",
    "share_with_displayname": "admin",
    "share_with_additional_info": null,
    "mail_send": 0,
    "attributes": null
   }
  ]
 }
}
```


Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI={oc-examples-server-url}
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl --user {oc-examples-username}:{oc-examples-password} \
   "$SERVER_URI/$API_PATH/shares?path=/Photos/Paris.jpg&reshares=true"
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
   'base_uri' =>
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1/',
]);

try {
   $response = $client->get('shares?path=/Photos/Paris.jpg&reshares=true',
[
      'auth' => ['your.username', 'your.password'],
      'debug' => true,
   ]);
   print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
   print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri =
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1/'
uri = URI("#{base_uri}/shares?path=/Photos/Paris.jpg&reshares=true")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Get.new uri
  req.basic_auth 'your.username', 'your.password'
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    serverUri :=
"https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1"
    username := "your.username"
    passwd := "your.password"

    client := &http.Client{}

    req, err := http.NewRequest("GET", fmt.Sprintf("%s/%s", serverUri,
"shares"), nil)
    if err != nil {
        log.Print(err)
        os.Exit(1)
    }

    // Add on some, relevant, query parameters
    q := req.URL.Query()
    q.Add("path", "/Photos/Paris.jpg")
    q.Add("reshares", "true")
    req.URL.RawQuery = q.Encode()

    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Get Information About A Known Share

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending/<share_id>
Method	GET

Supported Attributes

Attribute	Type	Description
share_id	int	The share’s unique id

Response Status Codes

Code	Description
100	Successful
404	Share doesn’t exist

Code Examples

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI=https://your.owncloud.install.com/owncloud
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl --user your.username:your.password "$SERVER_URI/
$API_PATH/shares/115464"
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
   'base_uri' =>
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1/',
]);

try {
   $response = $client->get('shares/115464', [
      'auth' => ['your.username', 'your.password'],
      'debug' => true,
   ]);
   print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
   print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri =
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1/'
uri = URI("#{base_uri}/shares/115464")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Get.new uri
  req.basic_auth 'your.username', 'your.password'
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    serverUri :=
"https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1"
    username := "your.username"
    passwd := "your.password"

    client := &http.Client{}

    req, err := http.NewRequest("GET", fmt.Sprintf("%s/%s", serverUri,
"shares/115464"), nil)
    if err != nil {
        log.Print(err)
        os.Exit(1)
    }

    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Kotlin

```
package main

import okhttp3.Credentials
import okhttp3.OkHttpClient
import okhttp3.Request
import java.io.IOException

fun main(args: Array<String>) {
    val ownCloudDomain = "your.owncloud.domain.com/owncloud"
    var client = OkHttpClient()
    val credentials = Credentials.basic("your.username",
"your.password");

    var builder = Request.Builder()
            .url
("https://$ownCloudDomain/ocs/v1.php/apps/files_sharing/api/v1/shares/<sh
are_id>'")
            .header("Authorization", credentials)
            .build()

    try {
        var response = client.newCall(builder).execute()

        when {
            response.isSuccessful -> println(
                    "Request was successful. Response was:
${response.body()?.string()}"
            )
            else -> println("Request was not successful.")
        }
    } catch (e: IOException) {
        println("Request failed. Reason: ${e.toString()}")
    }
}
```

Java

```
import okhttp3.Credentials;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

import java.io.IOException;

public class GetShareInfo {
  OkHttpClient client = new OkHttpClient();

  String run(String url, String credentials) throws IOException {
    Request request = new Request.Builder()
        .url(url)
        .header("Authorization", credentials)
        .build();

    try (Response response = client.newCall(request).execute()) {
      if (response.isSuccessful()) {
        String responseBody = (response.body().string() != null) ?
response.body().string() : "empty";
        return "Request was successful. Response was: " + responseBody;
      }

    } catch (IOException e) {
      return "Request was not successful. Reason: " + e.toString();
    }

    return "Request was not successful.";
  }

  public static void main(String[] args) throws IOException {
    GetShareInfo info = new GetShareInfo();

    String credentials = Credentials.basic("your.username",
"your.password");
    String ownCloudDomain = "your.owncloud.domain.com/owncloud";
    String url = "https://" + ownCloudDomain +
"/ocs/v1.php/apps/files_sharing/api/v1/shares/<share_id>'";

    String response = info.run(url, credentials);
    System.out.println(response);
  }
}
```

The Java and Kotlin examples use the square/okhttp library.

Example Response Payloads

Success

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data>
  <element>
   <id>115464</id>
   <share_type>6</share_type>
   <uid_owner>auser</uid_owner>
   <displayname_owner>A User</displayname_owner>
   <permissions>3</permissions>
   <stime>1481535991</stime>
   <parent/>
   <expiration/>
   <token>l5h8JYPb455oFkv</token>
   <uid_file_owner>auser</uid_file_owner>
   <displayname_file_owner>A User</displayname_file_owner>
   <path>/ownCloud Manual.pdf</path>
   <item_type>file</item_type>
   <mimetype>application/pdf</mimetype>
   <storage_id>home::auser</storage_id>
   <storage>993</storage>
   <item_source>3994491</item_source>
   <file_source>3994491</file_source>
   <file_parent>3994484</file_parent>
   <file_target></file_target>
   <share_with>user@example.com</share_with>

<share_with_displayname>user@example.com</share_with_displayname>
   <name>ownCloud Manual</name>
   <mail_send>0</mail_send>
  </element>
 </data>
</ocs>
```

Failure

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <status>failure</status>
    <statuscode>404</statuscode>
    <message>wrong share ID, share doesn't exist.</message>
  </meta>
  <data/>
</ocs>
```

Response Attributes

For details about the elements in the XML response payload please refer to the Response Attributes section of the Create a New Share section below.

Accept a Pending Share

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending/<share_id>
Method	POST

Request Attributes

Attribute	Type	Description
share id	integer	the id of the pending share to accept. Pending share ids are available in the get all shares response.

Status Codes

Code	Description
200	• Pending share successfully accepted. • Share doesn’t exist.

Example Request Response Payloads

Success

Listing 5. Pending share was successfully accepted

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
  <totalitems></totalitems>
  <itemsperpage></itemsperpage>
 </meta>
 <data>
  <element>
   <id>1</id>
   <share_type>0</share_type>
   <uid_owner>testuser</uid_owner>
   <displayname_owner>test user</displayname_owner>
   <permissions>19</permissions>
   <stime>1564484858</stime>
   <parent/>
   <expiration/>
   <token/>
   <uid_file_owner>testuser</uid_file_owner>
   <displayname_file_owner>test user</displayname_file_owner>
   <state>0</state>
   <path>/ownCloud Manual.pdf</path>
   <item_type>file</item_type>
   <mimetype>application/pdf</mimetype>
   <storage_id>shared::/ownCloud Manual.pdf</storage_id>
   <storage>3</storage>
   <item_source>97</item_source>
   <file_source>97</file_source>
   <file_parent>6</file_parent>
   <file_target>/ownCloud Manual.pdf</file_target>
   <share_with>admin</share_with>
   <share_with_displayname>admin</share_with_displayname>
   <share_with_additional_info/>
   <mail_send>0</mail_send>
   <attributes/>
  </element>
 </data>
</ocs>
```

Failure

Listing 6. The share id does not exist.

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>404</statuscode>
  <message>Wrong share ID, share doesn't exist</message>
  <totalitems></totalitems>
  <itemsperpage></itemsperpage>
 </meta>
 <data/>
</ocs>
```

Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI={oc-examples-server-url}
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl -X POST \
    --user {oc-examples-username}:{oc-examples-password} \
    "$SERVER_URI/$API_PATH/shares/pending/<share_id>"
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
    'base_uri' => '{oc-examples-server-
url}/ocs/v1.php/apps/files_sharing/api/v1/',
]);

try {
    $response = $client->request(
        'POST',
        'shares/pending/1',
        [
            'auth' => ['{oc-examples-username}', '{oc-examples-password}'],
            'debug' => true,
        ]
    );
    print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
    print $e->getMessage();
}
```

Ruby

```
require 'net/http'
require 'uri'

base_uri = '{oc-examples-server-url}/ocs/v1.php/apps/files_sharing/api/v1/'
uri = URI("#{base_uri}/shares/pending/1")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Post.new uri
  req.basic_auth '{oc-examples-username}', '{oc-examples-password}'
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    serverUri := "{oc-examples-server-
url}/ocs/v1.php/apps/files_sharing/api/v1"
    username := "{oc-examples-username}"
    passwd := "{oc-examples-password}"

    client := &http.Client{}

    req, err := http.NewRequest("POST", fmt.Sprintf("%s/%s", serverUri,
"shares/pending/<share_id>"), nil)
    if err != nil {
        log.Print(err)
        os.Exit(1)
    }

    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Decline a Pending Share

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending/<share_id>
Method	DELETE

Request Attributes

Attribute	Type	Description
share id	integer	the id of the pending share to decline. Pending share ids are available in the get all shares response.

Status Codes

Code	Description
200	• Pending share successfully declined (one or more times). • Share doesn’t exist.

Example Request Response Payloads

Listing 7. A pending share is successfully declined.

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
  <totalitems></totalitems>
  <itemsperpage></itemsperpage>
 </meta>
 <data/>
</ocs>
```

Listing 8. The share id does not exist or the pending share has already been declined.

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>404</statuscode>
  <message>Wrong share ID, share doesn't exist</message>
  <totalitems></totalitems>
  <itemsperpage></itemsperpage>
 </meta>
 <data/>
</ocs>
```

Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI={oc-examples-server-url}
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl -X DELETE \
    --user {oc-examples-username}:{oc-examples-password} \
    "$SERVER_URI/$API_PATH/shares/pending/<share_id>"
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
    'base_uri' => '{oc-examples-server-
url}/ocs/v1.php/apps/files_sharing/api/v1/',
]);

try {
    $response = $client->request(
        'DELETE',
        'shares/pending/<share_id>',
        [
            'auth' => ['{oc-examples-username}', '{oc-examples-password}'],
            'debug' => true,
        ]
    );
    print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
    print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri = '{oc-examples-server-url}/ocs/v1.php/apps/files_sharing/api/v1/'
uri = URI("#{base_uri}/shares/pending/<share_id>")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Delete.new uri
  req.basic_auth 'your.username', 'your.password'
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    serverUri := "{oc-examples-server-
url}/ocs/v1.php/apps/files_sharing/api/v1"
    username := "admin"
    passwd := "admin"

    client := &http.Client{}

    req, err := http.NewRequest("DELETE", fmt.Sprintf("%s/%s", serverUri,
"shares/pending/<share_id>"), nil)
    if err != nil {
        log.Print(err)
        os.Exit(1)
    }

    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Create A New Share

Share an existing file or folder with a user, a group, or as a public link.

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending
Method	POST

Function Arguments

Argument	Type	Description
name	string	A (human-readable) name for the share, which can be up to 64 characters in length.

Argument	Type	Description
path	string	The path to the file or folder which should be shared.
shareType	int	The type of the share. This can be one of: • 0 = user • 1 = group • 3 = public link • 6 = federated cloud share
shareWith	string	The user or group id with which the file should be shared.
publicUpload	boolean	Whether to allow public upload to a public link shared folder.
password	string	The password to protect the public link share with.
permissions	int	The permissions to set on the share. • 1 = read (default for public link shares); • 2 = update; • 4 = create; • 8 = delete; • 15 = read/write; • 16 = share; • 31 = All permissions.
expireDate	string	An expire date for public link shares. This argument expects a date string in the following format 'YYYY-MM-DD'.

Things to remember about public link shares

- Files will only ever have the **read** permission set
- Folders will have **read**, **update**, **create**, and **delete** set
- Public link shares **cannot** be shared with users and groups
- Public link shares are not available if public link sharing is disabled by the administrator

Mandatory Fields

shareType, path and shareWith are mandatory if shareType is set to 0 or 1

Returns

XML containing the share ID (int) of the newly created share

Status Codes

Code	Description
100	Successful
400	Unknown share type
403	Public upload was disabled by the admin
404	File or folder couldn’t be shared

Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI=https://your.owncloud.install.com/owncloud
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl --user your.username:your.password "$SERVER_URI/$API_PATH/shares" \
    --data
'path=/Photos/Paris.jpg&shareType=3&permissions=1&name=paris%20photo'
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
   'base_uri' =>
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1/',
]);

try {
   $response = $client->post('shares', [
      'auth' => ['your.username', 'your.password'],
      'debug' => true,
      'form_params' => [
         'path' => 'Photos/Paris.jpg',
         'shareType' => 3,
         'permissions' => 1
      ]
   ]);
   print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
   print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri =
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1'
uri = URI("#{base_uri}/shares")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Get.new uri
  req.basic_auth 'your.username', 'your.password'
  res = http.request req

  puts res.body
end
```

Go

```
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "net/url"
    "strconv"
    "strings"
)

func main() {
    serverUri :=
"https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1"
    username := "your.username"
    passwd := "your.password"

    client := &http.Client{}

    // Set the form POST body
    form := url.Values{}
    form.Add("path", "/Photos/Paris.jpg")
    form.Add("shareType", "3")
    form.Add("permissions", "1")

    // Build the core request object
    req, _ := http.NewRequest(
        "POST",
        fmt.Sprintf("%s/%s", serverUri, "shares"),
        strings.NewReader(form.Encode()),
    )
    req.Header.Add("Content-Type", "application/x-www-form-urlencoded")
    req.Header.Add("Content-Length", strconv.Itoa(len(form.Encode())))
    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Example Request Response Payloads

Failure

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>400</statuscode>
  <message>unknown share type</message>
 </meta>
 <data/>
</ocs>
```

Success

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data>
  <id>115468</id>
  <share_type>3</share_type>
  <uid_owner>auser</uid_owner>
  <displayname_owner>A User</displayname_owner>
  <permissions>1</permissions>
  <stime>1481537775</stime>
  <parent/>
  <expiration/>
  <token>MMqyHrR0GTepo4B</token>
  <uid_file_owner>auser</uid_file_owner>
  <displayname_file_owner>A User</displayname_file_owner>
  <path>/Photos/Paris.jpg</path>
  <item_type>file</item_type>
  <mimetype>image/jpeg</mimetype>
  <storage_id>home::auser</storage_id>
  <storage>993</storage>
  <item_source>3994486</item_source>
  <file_source>3994486</file_source>
  <file_parent>3994485</file_parent>
  <file_target>/Shared/Paris.jpg</file_target>
  <share_with/>
  <share_with_displayname/>

<url>https://your.owncloud.install.com/owncloud/index.php/s/MMqyHrR0GTepo4B<
/url>
  <mail_send>0</mail_send>
  <name>paris photo</name>
 </data>
</ocs>
```

Response Attributes

Argument	Type	Description
id	int	The share’s unique id.

Argument	Type	Description
share_type	int	The share's type. This can be one of: • 0 = user • 1 = group • 3 = public link • 6 = federated cloud share
uid_owner	string	The username of the owner of the share.
displayname_owner	string	The display name of the owner of the share.
permissions	octal a	The permission attribute set on the file. Options are: * 1 = Read * 2 = Update * 4 = Create * 8 = Delete * 16 = Share * 31 = All permissions The default is 31, and for public link shares is 1.
stime	int	The UNIX timestamp when the share was created.
parent	int	The UNIX timestamp when the share was created.
expiration	string	The UNIX timestamp when the share expires.
token	string	The public link to the item being shared.
uid_file_owner	string	The unique id of the user that owns the file or folder being shared.
displayname_file_owner	string	The display name of the user that owns the file or folder being shared.
path	string	The path to the shared file or folder.
item_type	string	The type of the object being shared. This can be one of file or folder.
mimetype	string	The RFC-compliant mimetype of the file.
storage_id	string	
storage	int	
item_source	int	The unique node id of the item being shared.
file_source	int	The unique node id of the item being shared. For legacy reasons item_source and file_source attributes have the same value.
file_parent	int	The unique node id of the parent node of the item being shared.
file_target	int	The name of the shared file.
share_with	string	The uid of the receiver of the file. This is either a GID (group id) if it is being shared with a group or a UID (user id) if the share is shared with a user.
share_with_displayname	string	The display name of the receiver of the file.
url	string	

Argument	Type	Description
mail_send	int	Whether the recipient was notified, by mail, about the share being shared with them.
name	string	A (human-readable) name for the share, which can be up to 64 characters in length

Delete A Share

Remove the given share.

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending/<share_id>
Method	DELETE

Attribute	Type	Description
share_id	int	The share's unique id

Status Codes

Code	Description
100	Successful
404	Share couldn't be deleted

Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI=https://your.owncloud.install.com/owncloud
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl --user your.username:your.password "$SERVER_URI/
$API_PATH/shares/115470" \
    --request DELETE
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
   'base_uri' =>
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/ap
i/v1/',
]);

try {
   $response = $client->delete('shares/115468', [
      'auth' => ['your.username', 'your.password'],
      'debug' => true,
   ]);
   print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
   print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri =
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1'
uri = URI("#{base_uri}/shares/115468")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Delete.new uri
  req.basic_auth 'your.username', 'your.password'
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

func main() {
    serverUri :=
"https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1"
    username := "your.username"
    passwd := "your.password"

    client := &http.Client{}

    // Build the core request object
    req, _ := http.NewRequest(
        "DELETE",
        fmt.Sprintf("%s/%s", serverUri, "shares/115470"),
        nil,
    )
    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Example Request Response Payloads

Failure

```xml
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data/>
</ocs>
```

Success

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>404</statuscode>
  <message>wrong share ID, share doesn't exist.</message>
 </meta>
 <data/>
</ocs>
```

Update Share

Update a given share. Only one value can be updated per request.

Endpoint	/ocs/v1.php/apps/files_sharing/api/v1/shares/pending/<share_id>	Method

Request Arguments

Argument	Type	Description
name	string	A (human-readable) name for the share, which can
		be up to 64 characters in length
share_id	int	The share's unique id
permissions	int	Update permissions
		(see the create share section above)
password	string	Updated password for a public link share
publicUpload	boolean	Enable (true) / disable (false)
		public upload for public link shares.
expireDate	string	Set an expire date for public link shares.
		This argument expects a well-formatted date string,
		such as: `YYYY-MM-DD'

Only one of the update parameters can be specified at once.

Status Codes

Code	Description
100	Successful
400	Wrong or no update parameter given
403	Public upload disabled by the admin
404	Couldn’t update share

Code Example

Curl

```
#!/bin/bash

##
## Variable Declaration
##
SERVER_URI=https://your.owncloud.install.com/owncloud
API_PATH=ocs/v1.php/apps/files_sharing/api/v1

curl --user your.username:your.password "$SERVER_URI/
$API_PATH/shares/115470" \
    --request PUT \
    --data 'expireDate=2017-01-02&name=paris%20photo'
```

PHP

```
<?php

use GuzzleHttp\Client;

require_once ('vendor/autoload.php');

// Configure the basic client
$client = new Client([
   'base_uri' =>
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/ap
i/v1/',
]);

try {
   $response = $client->put('shares/115470', [
      'auth' => ['your.username', 'your.password'],
      'debug' => true,
      'form_params' => [
         'expireDate' => '2017-01-01'
      ]
   ]);
   print $response->getBody()->getContents();
} catch (\GuzzleHttp\Exception\ClientException $e) {
   print $e->getMessage();
}
```

Ruby

```ruby
require 'net/http'
require 'uri'

base_uri =
'https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1'
uri = URI("#{base_uri}/shares/115470")

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do
|http|
  req = Net::HTTP::Put.new uri
  req.basic_auth 'your.username', 'your.password'
  req.set_form_data('expireDate' => '2017-01-03')
  res = http.request req

  puts res.body
end
```

Go

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "net/url"
    "strconv"
    "strings"
)

func main() {
    serverUri :=
"https://your.owncloud.install.com/owncloud/ocs/v1.php/apps/files_sharing/api/v1"
    username := "your.username"
    passwd := "your.password"

    client := &http.Client{}

    // Set the form POST body
    form := url.Values{}
    form.Add("expireDate", "2017-01-03")

    // Build the core request object
    req, _ := http.NewRequest(
        "PUT",
        fmt.Sprintf("%s/%s", serverUri, "shares/115470"),
        strings.NewReader(form.Encode()),
    )
    req.Header.Add("Content-Type", "application/x-www-form-urlencoded")
    req.Header.Add("Content-Length", strconv.Itoa(len(form.Encode())))
    req.SetBasicAuth(username, passwd)

    resp, err := client.Do(req)
    if err != nil {
        log.Fatal(err)
    }

    bodyText, err := ioutil.ReadAll(resp.Body)
    fmt.Println(string(bodyText))
}
```

Example Request Response Payloads

Failure

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>failure</status>
  <statuscode>400</statuscode>
  <message>can't change permission for public link share</message>
 </meta>
 <data/>
</ocs>
```

Success

```
<?xml version="1.0"?>
<ocs>
 <meta>
  <status>ok</status>
  <statuscode>100</statuscode>
  <message/>
 </meta>
 <data>
  <id>115470</id>
  <share_type>3</share_type>
  <uid_owner>auser</uid_owner>
  <displayname_owner>A User</displayname_owner>
  <permissions>1</permissions>
  <stime>1481552410</stime>
  <parent/>
  <expiration>2017-01-01 00:00:00</expiration>
  <token>11CUiVe0l7iaIwM</token>
  <uid_file_owner>auser</uid_file_owner>
  <displayname_file_owner>A User</displayname_file_owner>
  <path>/Photos/Paris.jpg</path>
  <item_type>file</item_type>
  <mimetype>image/jpeg</mimetype>
  <storage_id>home::auser</storage_id>
  <storage>993</storage>
  <item_source>3994486</item_source>
  <file_source>3994486</file_source>
  <file_parent>3994485</file_parent>
  <file_target>/Shared/Paris.jpg</file_target>
  <share_with/>
  <share_with_displayname/>
  <url>
https://your.owncloud.install.com/owncloud/index.php/s/11CUiVe0l7iaIwM</url>
  <mail_send>0</mail_send>
  <name>paris photo</name>
 </data>
</ocs>
```

Both the sending and the receiving instance need to have federated cloud sharing enabled and configured. See Configuring Federated Cloud Sharing.

Create A New Federated Cloud Share

Creating a federated cloud share can be done via the local share endpoint, using (int) 6 as a shareType and the Federated Cloud ID of the share recipient as shareWith. See Create a new Share for more information.

List Accepted Federated Cloud Shares

Get all federated cloud shares the user has accepted.

- Syntax: /remote_shares
- Method: GET

Returns

XML with all accepted federated cloud shares

Status Codes

Code	Description
100	Successful

Get Information About A Known Federated Cloud Share

Get information about a given received federated cloud share that was sent from a remote instance.

- Syntax: /remote_shares/<share_id>
- Method: GET

Attribute	Type	Description
share_id	int	The share id as listed in the id field
		in the remote_shares list

Returns

XML with the share information

Status Codes

Code	Description
100	Successful
404	Share doesn’t exist

Delete An Accepted Federated Cloud Share

Locally delete a received federated cloud share that was sent from a remote instance.

- Syntax: /remote_shares/<share_id>

- Method: DELETE

Attribute	Type	Description
share_id	int	The share id as listed in the id field
		in the remote_shares list

Status Codes

Code	Description
100	Successful
404	Share doesn't exist

List Pending Federated Cloud Shares

Get all pending federated cloud shares the user has received.

- Syntax: /remote_shares/pending
- Method: GET

Returns

XML with all pending federated cloud shares

Status Codes

Code	Description
100	Successful
404	Share doesn't exist

Accept a Pending Federated Cloud Share

Locally accept a received federated cloud share that was sent from a remote instance.

- Syntax: /remote_shares/pending/**<share_id>**
- Method: POST

Attribute	Type	Description
share_id	int	The share id as listed in the id field
		in the remote_shares/pending list

Status Codes

Code	Description
100	Successful
404	Share doesn't exist

Decline a Pending Federated Cloud Share

Locally decline a received federated cloud share that was sent from a remote instance.

- Syntax: /remote_shares/pending/<share_id>
- Method: DELETE

Attribute	Type	Description
share_id	int	The share id as listed in the id field
		in the remote_shares/pending list

Status Codes

Code	Description
100	Successful
404	Share doesn't exist

OCS TOTP (Time-based One-time Password) Validation API

Introduction

The OCS TOTP (Time-based One-time Password) Validation API allows administrator users to validate if a TOTP is valid.

Only admin accounts can use this API.

When 2FA (Two-Factor Authentication) is activated on an account, authorization with a username and password is not possible. Requests must authenticate via app passwords.

Prerequisites

This API requires the 2-Factor Authentication app to be installed and enabled.

Validate TOTP

- Path: ocs/v1.php/apps/twofactor_totp/api/v1/validate/<userid>/<totp>
- Method: GET

Request Parameters

Attribute	Type	Description
userid	string	The user id of the user to validate the TOTP for.
totp	string	The TOTP to validate.

Code Example

Curl

```
#!/usr/bin/env bash

USERNAME=admin
PASSWORD={oc-examples-password}
API_PATH="ocs/v1.php/apps/twofactor_totp/api/v1/validate/<userid>/<totp>
"
SERVER_URI="{oc-examples-server-url}"

curl '$SERVER_URI/$API_PATH/' \
  --user "${USERNAME}:${PASSWORD}"
```

Returns

The request returns either an XML (the default) or a JSON response, along with an HTTP 200 OK status code, which show whether:

1. The TOTP is valid
2. The TOTP is invalid
3. The user was not found

The status of the TOTP is located in the ocs/data/result element. If the user was not found, then:

1. ocs/meta/status will be set to failure.
2. ocs/meta/statuscode will be set to 404.

Example Responses

TOTP Is Valid

JSON


```
{
   "ocs": {
      "meta": {
         "status": "ok",
         "statuscode": 100,
         "message": "OK",
         "totalitems": "",
         "itemsperpage": ""
      },
      "data": {
         "result": true
      }
   }
}
```


XML


```
{
   "ocs": {
      "meta": {
         "status": "ok",
         "statuscode": 100,
         "message": "OK",
         "totalitems": "",
         "itemsperpage": ""
      },
      "data": {
         "result": true
      }
   }
}
```


TOTP Is Not Valid

JSON

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <status>ok</status>
    <statuscode>100</statuscode>
    <message>OK</message>
    <totalitems></totalitems>
    <itemsperpage></itemsperpage>
  </meta>
  <data>
    <result>1</result>
  </data>
</ocs>
```

XML

```
<?xml version="1.0"?>
<ocs>
  <meta>
    <status>ok</status>
    <statuscode>100</statuscode>
    <message>OK</message>
    <totalitems></totalitems>
    <itemsperpage></itemsperpage>
  </meta>
  <data>
    <result>1</result>
  </data>
</ocs>
```

User or Secret Not Found

JSON

```
{
   "ocs": {
      "meta": {
         "status": "failure",
         "statuscode": 404,
         "message": "OK",
         "totalitems": "",
         "itemsperpage": ""
      },
      "data": {
         "result": false
      }
   }
}
```

XML

```
<?xml version="1.0"?>
<ocs>
   <meta>
      <status>failure</status>
      <statuscode>404</statuscode>
      <message>OK</message>
      <totalitems></totalitems>
      <itemsperpage></itemsperpage>
   </meta>
   <data>
      <result></result>
   </data>
</ocs>
```

Unresolved directive in modules/developer_manual/pages/webdav_api/index.adoc - include::admin_manual:/drone/src/modules/developer_manual/pages/_partials/section_page.adoc[]

Comments API

Introduction

The comments API allows the following functionalities for files and folders stored in ownCloud.

It provides all of the functionality available through the UI and from the command-line.

List Comments

Request Path	Method	Content Type
remote.php/dav/comments/files/<fileid>	PROPFIND	text/xml

To retrieve a list of all comments, whether, for a file or folder, you need to make an authenticated PROPFIND request, and supply it with the path to the file or folder that you want to retrieve the comments of, as in the example below.

```
curl --silent -u username:password \
  -X PROPFIND \
  -H "Content-Type: text/xml" \
  'http://localhost/remote.php/dav/comments/files/4' | xmllint --format -
```

The response payload will look similar to the example below. It will contain a list of d:response elements, one for each comment attached to the file specified.

The example above uses xmllint, available in the libxml2 package to make the response easier to read.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/dav/comments/files/4/4</d:href>
    <d:propstat>
      <d:prop>
        <d:resourcetype/>
        <oc:id>1</oc:id>
        <oc:parentId>0</oc:parentId>
        <oc:topmostParentId>0</oc:topmostParentId>
        <oc:childrenCount>0</oc:childrenCount>
        <oc:message>Here is a comment.</oc:message>
        <oc:verb>comment</oc:verb>
        <oc:actorType>users</oc:actorType>
        <oc:actorId>admin</oc:actorId>
        <oc:creationDateTime>Tue, 16 May 2017 12:34:10
GMT</oc:creationDateTime>
        <oc:latestChildDateTime/>
        <oc:objectType>files</oc:objectType>
        <oc:objectId>4</oc:objectId>
        <oc:actorDisplayName>admin</oc:actorDisplayName>
        <oc:isUnread>false</oc:isUnread>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If you want to filter the information returned in the d:prop element of the XML response, you can supply a PROPFIND XML element in the body of the request method. The example below shows how to filter the information returned to just the oc:message element.

```
<?xml version="1.0" encoding="utf-8" ?>
<a:propfind xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
 <a:prop><oc:message/></a:prop>
</a:propfind>
```

To use it in the request, add the --data-binary switch, passing in the name of the file containing the PROPFIND XML element. I've called it report-propfind.xml in the example below.

```
curl --silent -u username:password \
  -X PROPFIND \
  -H "Content-Type: text/xml" \
   --data-binary "@report-propfind.xml" \
  'http://localhost/remote.php/dav/comments/files/4' | xmllint --format -
```

Create Comments

Request Path	Method	Content Type
remote.php/dav/comments/files/<fileid>	POST	application/json

To create a comment, you need to send an authenticated POST request with a JSON body containing the details of the comment to create. The example below shows how to create a comment on the file with the file id 4.

```
curl -u username:password \
  -X POST \
  -H "Content-Type: application/json" \
  --data-binary '{"message":"this is my
message","actorType":"users","verb":"comment"}' \
  "http://localhost/remote.php/dav/comments/files/4"
```

The available options are:

Parameter	Type	Description
actorType	String	The type of user who's adding the comment.
message	String	The comment's message text. It can be up to 1,000 characters in length.
verb	String	The type of comment to create, typically comment.

The comment is attributed to the user making the request.

To retrieve a file id, refer to the relevant section of the documentation.

Response

If the request is successful, there will be no response body returned. However, it will have an HTTP/1.1 201 Created status.

Update Comments

Request Path	Method	Content Type
remote.php/dav/comments/files/<fileid>/<commentid>	PROPPATCH	text/xml

To update an existing comment, you need to send an authenticated PROPPATCH request and provide a PROPFIND XML element in the body.

As with creating comments, we encourage you to store this in a separate file and use the --data-binary switch to include it in the request. This makes the information more maintainable.

Below is an example request, which will change the comment with the id of 4, on the file with the file id of 4.

```
curl -u username:password \
  -X PROPPATCH \
  -H "Content-Type: text/xml" \
  --data-binary "@update-comment.xml" \
  'http://localhost/remote.php/dav/comments/files/4/4' | xmllint --format -
```

Below is an example PROPPATCH element, which changes the message text but leaves the rest of the message unchanged.

```
<?xml version="1.0" encoding="utf-8" ?>
<a:propertyupdate xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
  <a:set>
    <a:prop>
      <oc:message>This is an updated message.</oc:message>
    </a:prop>
  </a:set>
</a:propertyupdate>
```

Response

Update comment requests will return the status: HTTP/1.1 207 Multi-Status, and an XML response similar to the example below. In it, you can see, in the d:href element the comment which was changed. In the d:status element, you can see if the update was successful or not.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/dav/comments/files/4/4</d:href>
    <d:propstat>
      <d:prop>
        <oc:message/>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If something goes wrong, you should receive a response similar to the following

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\BadRequest</s:exception>
  <s:message>This should never happen (famous last words)</s:message>
</d:error>
```

If the tag is not available, then you will receive the following response, along with an HTTP/1.1 404 Not Found status code.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotFound</s:exception>
  <s:message/>
</d:error>
```

Delete Comments

Request Path	Method	Content Type
remote.php/dav/comments/files/<fileid>/<commentid>	DELETE	text/plain

To delete a comment, send an authenticated DELETE request, specifying the path to the comment that you want to delete.

```
curl -u username:password -X DELETE
'http://localhost/remote.php/dav/comments/files/4/5'
```

If the comment was successfully deleted, no response body would be returned, but an HTTP/1.1 204 No Content status code will be returned. However, if the comment does not exist, then the following response will be returned, along with an HTTP/1.1 404 Not Found status code.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotFound</s:exception>
  <s:message/>
</d:error>
```

Group Management API

Custom Groups

List Groups

This endpoint returns a list of all custom groups.

URI	Request Type
PROPFIND	remote.php/dav/customgroups/groups/

```
curl --silent \
  -X PROPFIND \
  --data "@list-custom-groups.xml" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/remote.php/dav/customgroups/groups/' \
  | xmllint --format -
....

.list-custom-groups.xml
[source,xml]
....
<?xml version="1.0" encoding="UTF-8"?>
<oc:customgroups
    xmlns:a="DAV:"
    xmlns:oc="http://owncloud.org/ns">
    <a:prop>
        <a:getlastmodified />
        <a:getcontentlength/>
        <a:quota-used-bytes/>
        <a:quota-available-bytes/>
        <a:getetag/>
        <a:getcontenttype/>
    </a:prop>
</oc:customgroups>
....

Successful requests return two things:

. An XML payload.
. A status of `HTTP/1.1 207 Multi-Status`.

You can see an example of the XML payload below.
```

The XML payload contains a `response` element for each group.

[source,xml]
....
<?xml version="1.0"?>
<d:multistatus
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns"
   xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/customgroups/groups/</d:href>
  <d:propstat>
   <d:prop>
    <d:resourcetype>
     <d:collection/>
     <oc:customgroups-groups/>
    </d:resourcetype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
 <d:response>
  <d:href>/remote.php/dav/customgroups/groups/testgroup/</d:href>
  <d:propstat>
   <d:prop>
    <d:resourcetype>
     <d:collection/>
     <oc:customgroups-group/>
    </d:resourcetype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
  <d:propstat>
   <d:prop>
    <d:getlastmodified/>
    <d:getcontentlength/>
    <d:quota-used-bytes/>
    <d:quota-available-bytes/>
    <d:getetag/>
    <d:getcontenttype/>
   </d:prop>
   <d:status>HTTP/1.1 404 Not Found</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
....

==== No Results

If there are no custom groups, then a response similar to the following will be

returned.

[source,xml]
....
<?xml version="1.0"?>
<d:multistatus
    xmlns:d="DAV:"
    xmlns:s="http://sabredav.org/ns"
    xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/dav/customgroups/groups/</d:href>
    <d:propstat>
      <d:prop>
        <d:resourcetype>
          <d:collection/>
          <oc:customgroups-groups/>
        </d:resourcetype>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>

....

=== Rename Custom Group
:request_method: PROPPATCH
:request_data_file: rename-custom-group.xml
:request_path_suffix: <$groupId>

This endpoint allows a custom group to be renamed.

NOTE: Only group admins can rename the groups that they manage.

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`PROPFIND`
|`remote.php/dav/customgroups/groups/`
|===

[source,console,subs="attributes+"]

curl --silent \ -X PROPFIND \ --data "@list-custom-groups.xml" \ -u {oc-examples-username}:{oc-examples-password} \
'http://localhost/remote.php/dav/customgroups/groups/' \ | xmllint --format -

.{request_data_file}
[source,xml]

<?xml version="1.0" encoding="UTF-8"?> <a:propertyupdate xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns"> <a:prop> <oc:display-name>test_group</oc:display-name> </a:prop> </a:propertyupdate>

==== Responses

===== Success

A successful request will only return a status of `HTTP/1.1 204 No Content`.
No other information will be returned or displayed.

===== Failure

====== Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of `HTTP/1.1 401 Unauthorized` will be returned.

If the user making the request has insufficient privileges to make the request then a status of `HTTP/1.1 401 Unauthorized` will be returned, along with the following XML in the response's body:

[source,xml]

<?xml version="1.0" encoding="utf-8"?> <d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"> <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception> <s:message>No public access to this resource., Username or password was incorrect, Username or password was incorrect</s:message> </d:error>

====== Missing Group

If the specified group does not exist, then the following XML response body will be returned, along with an `HTTP/1.1 207 Multi-Status` status.

[source,xml]

<?xml version="1.0" encoding="utf-8"?> <d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"> <s:exception>Sabre\DAV\Exception\NotFound</s:exception> <s:message>Group with uri "testgroup" not found</s:message> </d:error>

```
=== Delete Group
:request_method: DELETE
:request_data_file:
:request_path_suffix: <$groupId>

This endpoint allows for a custom group to be deleted.

NOTE: Only group admins can delete a group.

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`{request_method}`
|`{request_base_path}/{request_path_suffix}`
|===


[source,console,subs="attributes+"]
----
curl --silent \
  -X {request_method} \
  --data "@{request_data_file}" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/{request_base_path}/{request_path_suffix}' \
  | xmllint --format -
```

Responses

Success

A successful request will only return a status of HTTP/1.1 204 No Content. No other information will be returned or displayed.

Failure

Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of HTTP/1.1 401 Unauthorized will be returned.

If the user making the request has insufficient privileges to make the request then a status of HTTP/1.1 401 Unauthorized will be returned, along with the following XML in the response's body:

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception>
  <s:message>No public access to this resource., Username or password was
incorrect, Username or password was incorrect</s:message>
</d:error>
```

Create Group

This endpoint allows for creating a custom group.

The group's creator automatically becomes the group's admin and its initial member.

URI	Request Type
MKCOL	remote.php/dav/customgroups/groups/<$groupId>

```
curl --silent \
  -X MKCOL \
  --data "@list-custom-groups.xml" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/remote.php/dav/customgroups/groups/&lt;$groupId&gt;' \
  | xmllint --format -
....

==== Responses

===== Success

A successful request will only return a status of `HTTP/1.1 201 Created`.
No other information will be returned or displayed.

===== Failure

====== Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of
`HTTP/1.1 401 Unauthorized` will be returned.

If the user making the request has insufficient privileges to make the request then a
status of `HTTP/1.1 401 Unauthorized` will be returned, along with the following
XML in the response's body:

[source,xml]
....
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception>
  <s:message>No public access to this resource., Username or password was
```

```
incorrect, Username or password was incorrect</s:message>
</d:error>
....


== Group Membership
:page-partial:
:request_base_path: /remote.php/dav/customgroups/users

// this page is included via groups.adoc
// some variables like request_base_path used in includes here are defined there

===  List Members
:request_method: PROPFIND
:request_data_file: list-custom-group-members.xml
:request_path_suffix:

This endpoint allows for listing all of the members in a custom group.

NOTE: Only group members can list a group's members. Other users will receive a
status of `HTTP/1.1 403 Forbidden`

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`MKCOL`
|`remote.php/dav/customgroups/groups/&lt;$groupId&gt;`
|===


[source,console,subs="attributes+"]
```

curl --silent \ -X MKCOL \ --data "@list-custom-groups.xml" \ -u {oc-examples-username}:{oc-examples-password} \ 'http://localhost/remote.php/dav/customgroups/groups/<$groupId>' \ | xmllint --format -

```
.{request_data_file}
[source,xml]
```

<?xml version="1.0" encoding="UTF-8"?> <a:propfind xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns"> <a:prop> <oc:role/> </a:prop> </a:propfind>

==== Responses

===== Success

Successful requests return two things:

. An XML payload.
. A status of `HTTP/1.1 207 Multi-Status`.

You can see an example of the XML payload below.

[source,xml]

<?xml version="1.0"?> <d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns" xmlns:oc="http://owncloud.org/ns"> <d:response> <d:href>/remote.php/dav/customgroups/groups/testgroup2/</d:href> <d:propstat> <d:prop> <oc:role/> </d:prop> <d:status>HTTP/1.1 404 Not Found</d:status> </d:propstat> </d:response> <d:response> <d:href>/remote.php/dav/customgroups/groups/testgroup2/admin</d:href> <d:propstat> <d:prop> <oc:role>admin</oc:role> </d:prop> <d:status>HTTP/1.1 200 OK</d:status> </d:propstat> </d:response> </d:multistatus>

===== Failure

====== Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of `HTTP/1.1 401 Unauthorized` will be returned.

If the user making the request has insufficient privileges to make the request then a status of `HTTP/1.1 401 Unauthorized` will be returned, along with the following XML in the response's body:

[source,xml]

<?xml version="1.0" encoding="utf-8"?> <d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"> <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception> <s:message>No public access to this resource., Username or password was incorrect, Username or password was incorrect</s:message> </d:error>

```
=== Add Member
:request_method: PUT
:request_data_file:
:request_path_suffix: <$numericGroupId>/<$userId>

This endpoint allows for adding members to a custom group.

NOTE: Only group admins can add members.

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`{request_method}`
|`{request_base_path}/{request_path_suffix}`
|===


[source,console,subs="attributes+"]
----
curl --silent \
  -X {request_method} \
  --data "@{request_data_file}" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/{request_base_path}/{request_path_suffix}' \
  | xmllint --format -
```

Responses

Success

If the request succeeds, then only a HTTP/1.1 201 Created status will be returned.

Failure

Method Not Allowed

If the request was made using any other method than PUT, then an HTTP/1.1 405 Method Not Allowed status will be returned, along with the XML payload below:

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\MethodNotAllowed</s:exception>
  <s:message>Cannot create collections</s:message>
</d:error>
```

Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of HTTP/1.1 401 Unauthorized will be returned.

If the user making the request has insufficient privileges to make the request then a status of HTTP/1.1 401 Unauthorized will be returned, along with the following XML in the response's body:

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception>
  <s:message>No public access to this resource., Username or password was
incorrect, Username or password was incorrect</s:message>
</d:error>
```

Remove Member

This endpoint allows for removing members from a custom group.

Only group admins can remove members. Group admins cannot remove themselves if no other admin exists in the group. A group member can remove themselves using this API call.

URI	Request Type
DELETE	remote.php/dav/customgroups/groups/<$numericGroupId>/<$userId>

```
curl --silent \
  -X DELETE \
  --data "@" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/remote.php/dav/customgroups/groups/&lt;$numericGroupId&gt
;/&lt;$userId&gt;' \
  | xmllint --format -
....

==== Responses

===== Success

A successful request will only return a status of `HTTP/1.1 204 No Content`.
No other information will be returned or displayed.

===== Failure

====== Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of
`HTTP/1.1 401 Unauthorized` will be returned.

If the user making the request has insufficient privileges to make the request then a
status of `HTTP/1.1 401 Unauthorized` will be returned, along with the following
XML in the response's body:
```

[source,xml]
....
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception>
  <s:message>No public access to this resource., Username or password was incorrect, Username or password was incorrect</s:message>
</d:error>
....

=== Change Admin Role of a Member
:request_method: PROPPATCH
:request_data_file:
:request_path_suffix: <$numericGroupId>/<$userId>

This endpoint allows for changing the admin role of an existing member of the group.

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`DELETE`
|`remote.php/dav/customgroups/groups/<$numericGroupId>/<$userId>`
|===

[source,console,subs="attributes+"]

curl --silent \ -X DELETE \ --data "@" \ -u {oc-examples-username}:{oc-examples-password} \
'http://localhost/remote.php/dav/customgroups/groups/<$numericGroupId>/<$userId>' \ | xmllint --format -

==== Responses

===== Success

===== Failure

====== Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of `HTTP/1.1 401 Unauthorized` will be returned.

If the user making the request has insufficient privileges to make the request then a status of `HTTP/1.1 401 Unauthorized` will be returned, along with the following XML in the response's body:

[source,xml]

<?xml version="1.0" encoding="utf-8"?> <d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"> <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception> <s:message>No public access to this resource., Username or password was incorrect, Username or password was incorrect</s:message> </d:error>

=== List Group Memberships of a Given User
:request_method: PROPFIND
:request_data_file:
:request_base_path: /remote.php/dav/customgroups/users
:request_path_suffix: <$userId>/<$membership>

This endpoint lists the groups that a user is a member of.

[cols="25%,75%",options="header,autowidth"]
|===
|URI
|Request Type

|`{request_method}`
|`{request_base_path}/{request_path_suffix}`
|===

[source,console,subs="attributes+"]
----
curl --silent \
  -X {request_method} \
  --data "@{request_data_file}" \
  -u {oc-examples-username}:{oc-examples-password} \
  'http://localhost/{request_base_path}/{request_path_suffix}' \
  | xmllint --format -

Responses

Success

Successful requests return two things:

1. An XML payload.
2. A status of HTTP/1.1 207 Multi-Status.

You can see an example of the XML payload below.

```
<?xml version="1.0"?>
<d:multistatus
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns"
   xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/customgroups/users/settermjd/</d:href>
  <d:propstat>
   <d:prop>
    <d:resourcetype>
     <d:collection/>
     <oc:customgroups-groups/>
    </d:resourcetype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
 <d:response>
  <d:href>/remote.php/dav/customgroups/users/settermjd/testgroup2/</d:href>
  <d:propstat>
   <d:prop>
    <d:resourcetype>
     <d:collection/>
     <oc:customgroups-group/>
    </d:resourcetype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
  <d:propstat>
   <d:prop>
    <d:getlastmodified/>
    <d:getcontentlength/>
    <d:quota-used-bytes/>
    <d:quota-available-bytes/>
    <d:getetag/>
    <d:getcontenttype/>
   </d:prop>
   <d:status>HTTP/1.1 404 Not Found</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
```

Failure

Insufficient Privileges or the User is not Authorized

If the user making the request that only and admin can perform, then a status of HTTP/1.1 401 Unauthorized will be returned.

If the user making the request has insufficient privileges to make the request then a

status of HTTP/1.1 401 Unauthorized will be returned, along with the following XML in the response's body:

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotAuthenticated</s:exception>
  <s:message>No public access to this resource., Username or password was incorrect, Username or password was incorrect</s:message>
</d:error>
```

Files Versions

Introduction

The files versions API allows for retrieving the following information about files stored in ownCloud.

List File Versions

Request Path	Method	Content Type
/remote.php/dav/meta/$fileid/v	PROPFIND	text/xml

If the file is not found, then the following response will be returned, with an HTTP/1.1 404 Not Found status:

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotFound</s:exception>
  <s:message>File not found: meta in 'root'</s:message>
</d:error>
```

Restore Another Version of a File

Request Path	Method	Content Type
/remote.php/dav/meta/$fileid/v/$versionid /remote.php/dav/files/$user/foo/bar	COPY	text/xml

The response payload will look similar to the example below.

Tags API

Introduction

The tags API provides extensive support for managing tags within ownCloud. In short, it provides all of the functionality available through the UI, from the command-line.

List Tags

Request Path	Method	Content Type
remote.php/dav/systemtags	PROPFIND	text/plain

To retrieve a list of all tags, stored in your ownCloud installation, you need to make an authenticated PROPFIND request, as in the example below.

```
curl --silent -u username:password \
  -X PROPFIND \
  'http://localhost/remote.php/dav/systemtags' | xmllint --format -
```

The curl examples use xmllint, available in the libxml2 package, to make the response easier to read.

This request will return an XML response similar to this example and a status of: HTTP/1.1 207 Multi-Status.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/systemtags/2</d:href>
  <d:propstat>
   <d:prop>
    <d:resourcetype/>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
```

Note that it does not return very much, just the href and status properties. If you want to retrieve more detailed information, you need to supply a PROPFIND element in the request body, containing all the properties that you want to retrieve in the response. The sample below, which for the purposes of this example we'll store in a file called report-propfind.xml, shows how to do so.

```
<?xml version="1.0" encoding="utf-8" ?>
<a:propfind xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
 <a:prop>
  <!-- Retrieve the display-name, user-visible, and user-assignable properties -->
  <oc:display-name/>
  <oc:user-visible/>
  <oc:user-assignable/>
  <oc:id/>
 </a:prop>
</a:propfind>
```

To use it in the request, add the --data-binary switch, passing in the name of the file

containing the PROPFIND XML element.

```
curl --silent -u username:password \
  -X PROPFIND \
  -H "Content-Type: text/xml" \
   --data-binary "@report-propfind.xml" \
  'http://localhost/remote.php/dav/systemtags' | xmllint --format -
```

We encourage you to store this in a separate file and use the --data-binary switch to include it in the request, instead of supplying the information in the command directly. This makes the information more maintainable.

Adding the PROPFIND XML element will cause the XML response to look similar to the following example.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/dav/systemtags/10</d:href>
    <d:propstat>
      <d:prop>
        <oc:display-name>file</oc:display-name>
        <oc:user-visible>true</oc:user-visible>
        <oc:id>10</oc:id>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
  <d:response>
    <d:href>/remote.php/dav/systemtags/9</d:href>
    <d:propstat>
      <d:prop>
        <oc:display-name>for</oc:display-name>
        <oc:user-visible>true</oc:user-visible>
        <oc:id>9</oc:id>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>
```

You can see that, along with the href and status elements, each element now contains the display-name, user-visible, and id elements. To clarify, display-name contains the visible tag name.

Create Tags

Request Path	Method	Content Type
remote.php/dav/systemtags	POST	application/json

To create a tag, you need to send an authenticated POST request with a JSON body containing the details of the tag to create. The example below shows how to create a tag with the name test5, which is visible to all users.

```
curl -u username:password \
  -X POST \
  -H "Content-Type: application/json" \
  --data-binary '{"name":"test5","userVisible":"true","userAssignable":"true"}' \
  "http://localhost/remote.php/dav/systemtags"
```

Available Parameters

Parameter	Type	Length	Required
name	string		yes
userVisible	boolean		no
userAssignable	boolean		no

Response

Regardless of success or failure, no response body is returned. However, if the tag is created successfully a status of HTTP/1.1 201 Created will be sent, and the location (and id) of the new tag will be available in the Content-Location header. For example: Content-Location: /remote.php/dav/systemtags/15. If a tag with the name supplied already exists a status of HTTP/1.1 409 Conflict will be sent.

Update Tags

Request Path	Method	Content Type
remote.php/dav/systemtags/<tagid>	PROPPATCH	text/xml

To update an existing tag, you need to send an authenticated PROPPATCH request and provide a PROPFIND XML element in the body. Below is an example request, which will change the tag with the id of 15.

```
curl -u username:password -X PROPPATCH \
  -H "Content-Type: text/xml" \
  --data-binary '@update-tag.xml' \
  "http://localhost/remote.php/dav/systemtags/15" | xmllint --format -
```

Below is an example PROPPATCH element, which changes the message text but leaves the rest of the message unchanged.

```
<?xml version="1.0" encoding="utf-8" ?>
<a:propertyupdate xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
 <a:set>
    <a:prop>
     <oc:display-name>This is an updated tag.</oc:display-name>
    </a:prop>
 </a:set>
</a:propertyupdate>
```

Response

If the update is successful, then an XML response body will be returned, which looks similar to the example below. In addition an HTTP/1.1 207 Multi-Status status will also be returned.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/systemtags/15</d:href>
  <d:propstat>
   <d:prop>
    <oc:name/>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
```

Delete Tags

Request Path	Method	Content Type
remote.php/dav/systemtags/<tagid>	DELETE	text/plain

To delete a tag, send an authenticated DELETE request, specifying the path to the tag that you want to delete.

```
curl -u username:password -X DELETE
'http://localhost/remote.php/dav/systemtags/15'
```

If the comment was successfully deleted, an HTTP/1.1 204 No Content status will be returned but with no response body. However, if the comment does not exist, then the following response will be returned, along with an HTTP/1.1 404 Not Found status.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\NotFound</s:exception>
  <s:message>Tag with id 15 not found</s:message>
</d:error>
```

Retrieve the Tag IDs and Metadata of a Given File

Request Path	Method	Content Type
remote.php/dav/systemtags-relations/files/<fileid>	PROPFIND	text/xml

To retrieve the tag ids and metadata of a given file, send an authenticated PROPFIND request, specifying the path to the file to retrieve the information from.

```
# Retrieve the details from file with id 4
curl -u username:password -X PROPFIND \
  -H "Content-Type: text/xml" \
  "http://localhost/remote.php/dav/systemtags-relations/files/4" | xmllint --format -
```

Response

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/dav/systemtags-relations/files/4/</d:href>
    <d:propstat>
      <d:prop>
        <d:resourcetype>
          <d:collection/>
        </d:resourcetype>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If more detailed information is desired, a PROPFIND element in the request body is required. The sample below, which for the purposes of this example we'll store in a file called report-propfind.xml will return the display-name, user-visible, user-assignable, and id values for each tag.

```
<?xml version="1.0" encoding="utf-8" ?>
<a:propfind xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
 <a:prop>
  <oc:display-name/>
  <oc:user-visible/>
  <oc:user-assignable/>
  <oc:id/>
 </a:prop>
</a:propfind>
```

To use it, as in previous examples, the --data-binary switch is required, as in the example below.

```
curl -u username:password -X PROPFIND \
  -H "Content-Type: text/xml" \
  --data-binary '@report-propfind.xml' \
  "http://localhost/remote.php/dav/systemtags-relations/files/4" | xmllint --format -
```

Below is an example of the response returned from this request:

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:cal="urn:ietf:params:xml:ns:caldav" xmlns:cs="http://calendarserver.org/ns/"
xmlns:card="urn:ietf:params:xml:ns:carddav" xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/systemtags-relations/files/4/2</d:href>
  <d:propstat>
   <d:prop>
    <oc:display-name>test</oc:display-name>
    <oc:user-visible>true</oc:user-visible>
    <oc:user-assignable>true</oc:user-assignable>
    <oc:id>2</oc:id>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
```

Assign a Tag to a File

Request Path	Method	Content Type
remote.php/dav/systemtags-relations/files/<fileid>/<tagid>	PUT	text/xml

To assign a tag to a file, send an authenticated PUT request specifying the path to the file to tag. Here is an example of how to do it using Curl.

```
curl -u username:password -X PUT \
  -H "Content-Type: text/xml" \
  "http://localhost/remote.php/dav/systemtags-relations/files/4/6"
```

Response

If the request is successful, no response body will be returned, but an HTTP/1.1 201 Created status will be returned. If the request is not successful, then either an HTTP/1.1 404 Not Found or an HTTP/1.1 409 Conflict status will be returned. A 404 status is returned if the file or folder doesn't exist. A 409 status is returned if the tag has already been assigned to that file or folder.

Unassign a Tag From a File

Request Path	Method	Content Type
remote.php/dav/systemtags-relations/files/<fileid>/<tagid>	DELETE	text/xml

To un-assign or remove a tag from a file, send an authenticated DELETE request specifying the path to the file and the tag to remove. Here is an example of how to do it using Curl.

```
curl --silent --verbose -u username:password -X DELETE \
  -H "Content-Type: text/xml" \
  "http://localhost/remote.php/dav/systemtags-relations/files/4/6"
```

Response

If the request is successful, no response body will be returned, but an HTTP/1.1 204 No Content status will be returned. If the request is not successful, likely because the tag was not assigned to the file or folder, then an HTTP/1.1 404 Not Found status will be returned.

Create and Assign a Tag at the Same Time

Request Path	Method	Content Type
remote.php/dav/systemtags-relations/files/<fileid>	POST	application/json

In addition to assigning existing tags to a file, you can also create a new tag and assign it to a file in one request. You do this by sending an authenticated POST request specifying the path to the file and a JSON body containing the details of the tag to create.

The new tag will be created and assigned, effectively, in one atomic operation. Here is an example of how to do it using Curl.

```
curl --silent --verbose -u username:password -X POST \
  -H "Content-Type: application/json" \
  --data-binary '{"name":"variabletag","userVisible":"true","userAssignable":"true"}'
\
  "http://localhost/remote.php/dav/systemtags-relations/files/4"
```

If the request is successful, no response body will be returned, but an HTTP/1.1 201 Created status will be returned. If the request is not successful, likely because the tag already exists, then an HTTP/1.1 409 Conflict status will be returned.

Retrieve All Files Tagged with a Tag ID

Request Path	Method	Content Type
remote.php/webdav/	REPORT	text/xml

To retrieve all the files tagged with a given tag id send an authenticated REPORT request with a PROPFIND element in the request body containing the tag id to filter on and the list of properties to return.

The sample a PROPFIND element below, which for the purposes of this example we'll store in a file called report-propfind.xml, will return every tag property, and will filter on tag id 17.

```
<oc:filter-files  xmlns:d="DAV:" xmlns:oc="http://owncloud.org/ns">
   <d:prop>
      <d:getlastmodified />
      <d:getetag />
      <d:getcontenttype />
      <d:resourcetype />
      <oc:fileid />
      <oc:permissions />
      <oc:size />
      <d:getcontentlength />
      <oc:tags />
      <oc:favorite />
      <oc:comments-unread />
      <oc:owner-display-name />
      <oc:share-types />
   </d:prop>
   <oc:filter-rules>
      <oc:systemtag>17</oc:systemtag>
   </oc:filter-rules>
</oc:filter-files>
```

And here is an example of how to make the request using Curl.

```
curl --silent --verbose -u username:password -X REPORT \
  -H "Content-Type: text/xml" \
  --data-binary "@find-tags-by-file.xml" \
  "http://localhost/remote.php/webdav/" | xmllint --format -
```

Response

A successful response which you can see an example of below, along with a status of HTTP/1.1 207 Multi-Status will be returned.

```
<?xml version="1.0"?>
<d:multistatus xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns"
xmlns:oc="http://owncloud.org/ns">
  <d:response>
    <d:href>/remote.php/webdav/Photos/Squirrel.jpg</d:href>
    <d:propstat>
      <d:prop>
        <d:getlastmodified>Wed, 03 May 2017 11:05:49
GMT</d:getlastmodified>
        <d:getetag>"0169c644a1580687b346ef43315d5ac8"</d:getetag>
        <d:getcontenttype>image/jpeg</d:getcontenttype>
        <d:resourcetype/>
        <oc:fileid>6</oc:fileid>
        <oc:permissions>RDNVW</oc:permissions>
        <oc:size>233724</oc:size>
        <d:getcontentlength>233724</d:getcontentlength>
        <oc:tags/>
        <oc:favorite>0</oc:favorite>
        <oc:comments-unread>0</oc:comments-unread>
        <oc:owner-display-name>admin</oc:owner-display-name>
        <oc:share-types/>
      </d:prop>
      <d:status>HTTP/1.1 200 OK</d:status>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If the request was unsuccessful, likely because the tag specified didn't exist, then an HTTP/1.1 412 Precondition failed status will be returned, along with the following XML payload in the body of the response.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error xmlns:d="DAV:" xmlns:s="http://sabredav.org/ns">
  <s:exception>Sabre\DAV\Exception\PreconditionFailed</s:exception>
  <s:message>Cannot filter by non-existing tag</s:message>
</d:error>
```

Introduction

If you need to search for files, then you can use the WebDAV search API. The search API exposes two endpoints for finding files in a user's filesystem.

Search Files

The search-files report search through the available files in an ownCloud user's filesystem, based on a rudimentary filename pattern match.

By default, the report uses ownCloud's default search provider to power the search functionality. However, other search providers, such as search_elastic and search_lucene greatly enrich the ability to search, such as being able to search through file content, as well as by a file's name. When installed, they replace ownCloud's default search provider and the search API will automatically use them.

When using the default search provider, if you use the search string "*ownCloud*", files whose filename has "ownCloud" in it will be matched. However, if installed the search_elastic app, the report also retrieves files that have "*ownCloud*" in the file's contents.

Core Details

Request Path	Method	Content Type
remote.php/dav/files/<user>	REPORT	text/xml

The Request

An authenticated REPORT request needs to be made to search for all files stored in a user's ownCloud filesystem

Example Request

```
curl --silent \
  -X REPORT \
  --data "@supported.xml" \
  -u admin:admin \
  'http://localhost/remote.php/dav/files/admin' | xmllint --format -
```

The request must include a request body that includes the search pattern, and can also include a list of properties to return.

Example Request Bodies

Below, are several examples of XML response bodies.

Filtering By Filename Pattern

In the search element, specify the search pattern to filter the list of files to return.

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:search-files
  xmlns:a="DAV:"
  xmlns:oc="http://owncloud.org/ns">
  <oc:search>
    <oc:pattern>web</oc:pattern>
  </oc:search>
</oc:search-files>
```

Limiting The Number Of Results Returned

To limit the number of results returned, use the limit element of the search element, as in the following example. It will limit the maximum number of results returned to five.

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:search-files
  xmlns:a="DAV:"
  xmlns:oc="http://owncloud.org/ns">
  <oc:search>
    <oc:pattern>web</oc:pattern>
    <oc:limit>5</oc:limit>
  </oc:search>
</oc:search-files>
```

Reducing The File Properties Returned

However, if a specific list of properties is required for each file, then a prop element needs to be included in the response body, such as in the example below.

Table 1. Available File Properties

Property	Description	Namespace
id	The id of the file	http://owncloud.org/ns
permissions	The permissions set on the file	http://owncloud.org/ns
size	The file's size	http://owncloud.org/ns
owner-id	The id of the file owner	http://owncloud.org/ns
owner-display-name	The display name of the file owner	http://owncloud.org/ns
getlastmodified	The last modified date of the file	DAV
getetag	The file's ETag	DAV
getcontenttype	The file's content type.	DAV

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:search-files
  xmlns:a="DAV:"
  xmlns:oc="http://owncloud.org/ns">
  <a:prop>
    <oc:id/>
    <oc:downloadURL/>
    <oc:fileid/>
    <oc:permissions/>
    <oc:size/>
    <oc:owner-id/>
    <oc:owner-display-name/>
    <a:getlastmodified/>
    <a:getetag/>
    <a:getcontenttype/>
  </a:prop>
  <oc:search>
    <oc:pattern>site</oc:pattern>
  </oc:search>
</oc:search-files>
```

The example uses xmllint to make the response more readable. Xmllint is available in the libxml2 package.

The Response

Success

Successful requests return two things:

1. An XML payload.
2. A status of HTTP/1.1 207 Multi-Status.

You can see an example of the XML payload below. The XML payload contains a response element for each file. And each response element contains three items:

1. A link to the file (href).
2. The requested properties, along with their respective values (propstat).
3. The file's status (status).

Listing 9. Search Response

```
<?xml version="1.0"?>
<d:multistatus
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns"
   xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/files/admin/Test/Sub-test/Site-Plan.md</d:href>
  <d:propstat>
   <d:prop>
    <oc:id>00000065oc21s4c9iej2</oc:id>
    <oc:downloadURL/>
    <oc:fileid>65</oc:fileid>
    <oc:permissions>RDNVW</oc:permissions>
    <oc:size>423</oc:size>
    <oc:owner-id>admin</oc:owner-id>
    <oc:owner-display-name>admin</oc:owner-display-name>
    <d:getlastmodified>Fri, 28 Jul 2017 05:51:07 GMT</d:getlastmodified>
    <d:getetag>"0286fcdabf5b4f5ef84788d86c37e245"</d:getetag>
    <d:getcontenttype>text/markdown</d:getcontenttype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
```

Failure

If The Payload File Cannot Be Read Or Is Invalid XML

If the payload file cannot be read or is invalid XML, then the following XML response is sent, along with an HTTP/1.1 500 Internal Server Error status code.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns">
 <s:exception>Sabre\Xml\ParseException</s:exception>
 <s:message>This should never happen (famous last words)</s:message>
</d:error>
```

If a Non-Existent Property Is Requested

If a non-existent property is requested, then an additional propstat element is returned, as in the example below, which contains a list of the properties which were not available.

```
  <d:status>HTTP/1.1 200 OK</d:status>
</d:propstat>
<d:propstat>
  <d:prop>
    <oc:downloadUR/>
  </d:prop>
  <d:status>HTTP/1.1 404 Not Found</d:status>
```

Filter Files

The filter-files report allows for retrieving a list of files in an ownCloud user's filesystem, based on two criteria:

Core Details

Request Path	Method	Content Type
remote.php/dav/files/<user>	REPORT	text/xml

The Request

An authenticated REPORT request needs to be made to retrieve a list of all files stored in a user's ownCloud filesystem.

Example Request

```
curl --silent \
  -X REPORT \
  --data "@filter-files-criteria.xml" \
  -u admin:admin \
  'http://localhost/remote.php/dav/files/admin' | xmllint --format -
```

The request must include a request body that includes the rules to filter by. There are two filter rules which can be supplied; these are:

Rule	Description	Type	Accepted Values	Mandatory
favorite	Whether they've been marked as a favorite or not (mandatory)	integer	0,1	Yes
systemtag	The tags that have been assigned to them	integer	Any valid system tag. These can be retrieved by using the Tags API.	No

Example Request Bodies

Below, are several examples of the XML response bodies that can be sent with the request.

Minimal Request Body

In the search element, it specifies the search pattern to filter down the list of files to return in a successful resultset.

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:filter-files xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
   <oc:filter-rules>
      <oc:favorite>1</oc:favorite>
   </oc:filter-rules>
</oc:filter-files>
```

Limiting Returned File Properties

If only a specific list of properties is required for each file, then a prop element needs to be included in the response body, such as in the example below.

Table 2. Available File Properties

Property	Description	Namespace
id	The id of the file	http://owncloud.org/ns
permissions	The permissions set on the file	http://owncloud.org/ns
size	The file's size	http://owncloud.org/ns
owner-id	The id of the file owner	http://owncloud.org/ns
owner-display-name	The display name of the file owner	http://owncloud.org/ns
getlastmodified	The last modified date of the file	DAV
getetag	The file's ETag	DAV
getcontenttype	The file's content type.	DAV

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:filter-files xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
   <a:prop>
      <oc:fileid/>
      <oc:permissions/>
      <oc:size/>
      <oc:owner-id/>
      <oc:owner-display-name/>
      <a:getlastmodified/>
      <a:getetag/>
      <a:getcontenttype/>
   </a:prop>
   <oc:filter-rules>
      <oc:favorite>1</oc:favorite>
   </oc:filter-rules>
</oc:filter-files>
```

Filtering By Tag

Files can be filtered by those assigned specific tags. If this is required, then the systemtag element needs to be supplied, which contains a space-separated list of tag *ids* to filter by.

Tag ids can be retrieved by using the Tags API.

```
<?xml version="1.0" encoding="UTF-8"?>
<oc:filter-files xmlns:a="DAV:" xmlns:oc="http://owncloud.org/ns">
  <a:prop>
    <oc:fileid/>
    <oc:permissions/>
    <oc:size/>
    <oc:owner-id/>
    <oc:owner-display-name/>
    <a:getlastmodified/>
    <a:getetag/>
    <a:getcontenttype/>
  </a:prop>
  <oc:filter-rules>
    <oc:favorite>1</oc:favorite>
    <oc:systemtag>1</oc:systemtag>
  </oc:filter-rules>
</oc:filter-files>
```

The example uses xmllint to make the response more readable. Xmllint is available in the libxml2 package.

The Response

Success

Successful requests return two things:

1. An XML payload.
2. A status of HTTP/1.1 207 Multi-Status.

You can see an example of the XML payload below. The XML payload contains a response element for each file. And each response element contains three items:

1. A link to the file (href).
2. The requested properties, along with their respective values (propstat).
3. The file's status (status).

Listing 10. Example of a successful search response

```
<?xml version="1.0"?>
<d:multistatus
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns"
   xmlns:oc="http://owncloud.org/ns">
 <d:response>
  <d:href>/remote.php/dav/files/admin/welcome.txt</d:href>
  <d:propstat>
   <d:prop>
    <oc:fileid>28</oc:fileid>
    <oc:permissions>RDNVW</oc:permissions>
    <oc:size>163</oc:size>
    <oc:owner-id>admin</oc:owner-id>
    <oc:owner-display-name>admin</oc:owner-display-name>
    <d:getlastmodified>Mon, 05 Nov 2018 10:52:58
GMT</d:getlastmodified>
    <d:getetag>"91b08390250f5294390c4fc92b6b0138"</d:getetag>
    <d:getcontenttype>text/plain</d:getcontenttype>
   </d:prop>
   <d:status>HTTP/1.1 200 OK</d:status>
  </d:propstat>
 </d:response>
</d:multistatus>
```

Failure

If The Payload File Cannot Be Read Or Is Invalid XML

If the payload file cannot be read or is invalid XML, then the following XML response is sent, along with an HTTP/1.1 500 Internal Server Error status code.

```
<?xml version="1.0" encoding="utf-8"?>
<d:error
   xmlns:d="DAV:"
   xmlns:s="http://sabredav.org/ns">
 <s:exception>Sabre\Xml\ParseException</s:exception>
 <s:message>This should never happen (famous last words)</s:message>
</d:error>
```

If a Non-Existent Property Is Requested

If a non-existent property is requested, then an additional propstat element is returned, as in the example below, which contains a list of the properties which were not available.

```
  <d:status>HTTP/1.1 200 OK</d:status>
</d:propstat>
<d:propstat>
  <d:prop>
    <oc:downloadUR/>
  </d:prop>
  <d:status>HTTP/1.1 404 Not Found</d:status>
```

Introduction

ownCloud applications let you extend and build on the power of ownCloud, taking it in ways that work just for your specific use-case(s).

While not overly complex in nature, like any form of software development, it will take some time to become fully knowledgeable on the parts that make up and application, how they fit together, and how to make best use of them.

This section of the documentation's been designed to make that process as simple and as effective as possible, by both stepping you through the information in a tutorial-fashion, as well as providing you a significant amount of background technical knowledge.

You'll learn how an application works by building one. It won't do absolutely everything that you could possibly hope to cover. But it will teach you the ins and outs of building one, providing links to further information, which you can work through later.

Before you start developing an ownCloud application, please check that there isn't an application in the ownCloud Marketplace, or an official ownCloud app that already does what you need. If there is, we strongly encourage you to contribute to existing applications before investing the time to develop your own. Also, feel free to communicate your idea and plans to the user mailing list or developer mailing list so other contributors might join in.

Application Development - Fundamental Details

In this section, you will find the fundamental details for developing an ownCloud application.

Application Metadata

The appinfo/info.xml contains metadata about the application. In this section, you will find a complete example configuration, along with an explanation of what each of file's elements.

```
<?xml version="1.0"?>
<info>
   <!-- Mandatory fields -->
   <id>yourappname</id>
   <name>Your App</name>
   <description>Your application description</description>
   <version>1.0</version>
   <licence>AGPL</licence>
   <screenshot small-
```

```
thumbnail="https://raw.githubusercontent.com/foo/myapp/master/screenshots/thumb.png"
    >
https://raw.githubusercontent.com/foo/myapp/master/screenshots/big.png</screenshot>
  <!-- Category values available at: https://marketplace.owncloud.com/ajax/categories -->
  <category>A category for the application. </category>
  <summary>A summary of the application's purpose (max 90 chars)</summary>

  <types>
    <filesystem/>
  </types>

  <documentation>

<user>https://doc.owncloud.com/server/latest/user_manual/pim/contacts.html</user>

<admin>https://doc.owncloud.com/server/latest/admin_manual/configuration_server/occ_command.html?highlight=contact#dav-commands</admin>
      <developer>
https://github.com/owncloud/contacts/blob/master/README.md</developer>
  </documentation>

  <author>Your Name</author>
  <namespace>YourapplicationsNamespace</namespace>
  <website>https://owncloud.org</website>
  <bugs>https://github.com/owncloud/theapp/issues</bugs>
  <repository type="git">
https://github.com/owncloud/theapplication.git</repository>
  <ocsid>1234</ocsid>

  <dependencies>
    <php min-version="5.4" max-version="5.5"/>
    <database>sqlite</database>
    <database>mysql</database>
    <command os="linux">grep</command>
    <command os="windows">notepad.exe</command>
    <lib min-version="1.2">xml</lib>
    <lib max-version="2.0">intl</lib>
    <lib>curl</lib>
    <os>Linux</os>
    <owncloud min-version="6.0.4" max-version="8"/>
  </dependencies>

  <!-- For registering panels -->
  <settings>
    <admin>OCA\MyApp\Settings\Admin</admin>
```

```
    <personal>OCA\MyApp\Settings\Personal</personal>
  </settings>

  <!-- For registering settings sections -->
  <settings-sections>
    <admin>OCA\MyApp\Settings\AdminSection</admin>
    <personal>OCA\MyApp\Settings\PersonalSection</personal>
  </settings-sections>

  <!-- deprecated, but kept for reference -->
  <public>
    <file id="caldav">appinfo/caldav.php</file>
  </public>
  <remote>
    <file id="caldav">appinfo/caldav.php</file>
  </remote>
  <standalone />
  <default_enable />
  <shipped>true</shipped>
  <!-- end deprecated -->
</info>
```

id

Required. This field contains the internal application name, and has to be the same as the folder name of the application. This id needs to be unique in ownCloud, meaning no other application should have this id. This value also represents the URL your application is available on the marketplace.

name

Required. This is the human-readable name (or title) of the application that will be displayed in the application overview page.

description

Required. The description provides all the necessary information about the application, and is shown in the application overview page. Don't get lost in technical details, focus on the benefits which the application offers. You can use markdown to format the description.

Max. 4000 characters.

version

This sets the version of your application.

licence

Required. The sets the application's license. This license must be compatible with the AGPL and **must not be proprietary**.

Two good examples are:

- AGPL 3 (recommended)

- MIT

If a proprietary/non-AGPL compatible license must be used, then you have to use the ownCloud Enterprise Edition.

author

Required. The name of the application's author or authors.

namespace

Required if routes.php returns an array. For example, if your application is namespaced, e.g., \\OCA\\MyApp\\Controller\\PageController, then the required namespace value is MyApp. If a namespace is not provided, the application tries to default to the first letter upper-cased application id, e.g., myapp would be tried under Myapp.

category

The ownCloud Marketplace category where you want to publish the application. The following categories are available:

Category Name	Value to Use
Automation	automation
Collaboration	collaboration
Customization	customization
External plugins	external-plugins
Games	games
Integration	integration
Multimedia	multimedia
Productivity	productivity
Security	security
Storage	storage
Tools	tools

For publishing themes the category tag must be present — *but empty* — as in the example below.

```
<category></category>
```

summary

Required. Provide a short application description (max. 90 chars). This gets displayed below the product title and on the product tiles. It is mandatory since ownCloud 10.0.0.

types

ownCloud supports five types. These are:

- prelogin: applications which need to load on the login page

- filesystem: applications which provide filesystem functionality (e.g., file-sharing applications)
- authentication: applications which provide authentication backends
- logging: applications which implement a logging system
- prevent_group_restriction: applications which can not be enabled for specific groups (e.g., notifications app).

prevent_group_restriction was introduced with ownCloud 9.0. It can be used in earlier versions, but the functionality will be ignored.

Due to technical reasons applications of any type listed above can not be enabled for specific groups only.

documentation

Required. Link to *admin*, *user*, and *developer* documentation. Common places are: (where $name is the name of your app, e.g. $name=theapp)

```
$DOCUMENTATION_BASE = 'https://doc.owncloud.com';
$DOCUMENTATION_DEVELOPER =
$DOCUMENTATION_BASE.'/server/'.$VERSIONS_SERVER_MAJOR_DEV_DOCS.'/develo
per_manual/$name/';`
$DOCUMENTATION_ADMIN =
$DOCUMENTATION_BASE.'/server/'.$VERSIONS_SERVER_MAJOR_STABLE.'/admin_ma
nual/$name/';
$DOCUMENTATION_USER =
$DOCUMENTATION_BASE.'/server/'.$VERSIONS_SERVER_MAJOR_STABLE.'/user_man
ual/$name/';
```

These places are maintained at https://github.com/owncloud/documentation/. Another popular starting point for developer documentation is the README.md in GitHub.

website

Required. A link to the project's web page.

repository

Required. A link to the version control repository.

bugs

Required. A link to the bug tracker, if any.

Dependencies

All tags within the dependencies tag define a set of requirements which have to be fulfilled in order to operate properly. As soon as one of these requirements is not met the application cannot be installed.

php

Defines the minimum and the maximum version of PHP required to run this application.

database

Each supported database has to be listed here. Valid values are sqlite, mysql, pgsql, oci and mssql. In the future it will be possible to specify versions here as well. In case no database is specified it is assumed that all databases are supported.

command

Defines a command line tool to be available. With the attribute os the required operating system for this tool can be specified. Valid values for the os attribute are as returned by the php function php_uname.

lib

Defines a required PHP extension with a required minimum and/or maximum version. The names for the libraries have to match the result as returned by the php function get_loaded_extensions. The explicit version of an extension is read from phpversion - with some exception as to be read up in the code base

os

Defines the required target operating system the application can run on. Valid values are as returned by the php function php_uname.

owncloud

Defines the minimum and maximum versions of ownCloud core.

This will be mandatory from version 11 onwards.

Deprecated

The following sections are listed just for reference and should not be used because:

- **public/remote**: Use api instead because you'll have to use the external API, which is known to be buggy. It only works properly with GET/POST requests.
- **standalone/default_enable**: They tell core what do on setup, you will not be able to even activate your application if it has those entries.

This should be replaced by a config file inside core.

public

Used to provide a public interface (requires no login) for the application. The id is appended to the URL /owncloud/index.php/public. Example with id set to `calendar':

```
/owncloud/index.php/public/calendar
```

Also take a look at the external API.

remote

Same as public, but requires login. The id is appended to the URL /owncloud/index.php/remote. Example with id set to `calendar':

```
/owncloud/index.php/remote/calendar
```

Also take a look at the external API.

standalone

Can be set to true to indicate that this application is a web application. This can be used to tell GNOME Web for instance to treat this like a native application.

default_enable

Core applications only: Used to tell ownCloud to enable them after the installation.

shipped

Core applications only: Used to tell ownCloud that the application is in the standard release. Please note that if this attribute is set to FALSE or not set at all, every time you disable the application, all the files of the application itself will be *REMOVED* from the server!

The Classloader

The classloader is provided by ownCloud and loads all your classes automatically. The only thing left to include by yourself are 3rd party libraries. Those should be loaded in lib/AppInfo/Application.php.

PSR-4 Autoloading

Since ownCloud 9.1 there is a PSR-4 autoloader in place. The namespace \\OCA\\MyApp is mapped to /apps/myapp/lib/. Afterward, normal PSR-4 rules apply, so a folder is a namespace section in the same casing and the class name matches the file name.

If your appid can not be turned into the namespace by upper-casing the first character, you can specify it in your appinfo/info.xml by providing a field called namespace. The required namespace is the one which comes after the top level namespace OCA\\, e.g.: for OCA\\MyBeautifulApp\\Some\\OtherClass the needed namespace would be MyBeautifulApp and would be added to the info.xml in the following way:

```
<?xml version="1.0"?>
<info>
  <namespace>MyBeautifulApp</namespace>
  <!-- other options here ... -->
</info>
```

A second PSR-4 root is available when running tests. \\OCA\\MyApp\\Tests is thereby mapped to /apps/myapp/tests/.

Legacy Autoloading

The legacy classloader, deprecated since 9.1, is still in place and works like this:

- Take the full qualifier of a class

```
\OCA\MyApp\Controller\PageController
```

- If it starts with \\OCA, then include the file from the apps directory
- Cut off \\OCA

```
\MyApp\Controller\PageController
```

- Convert all characters to lowercase

```
\myapp\controller\pagecontroller
```

- Replace \ with /

```
/myapp/controller/pagecontroller
```

- Append .php

```
/myapp/controller/pagecontroller.php
```

- Prepend /apps because of the OCA namespace and include the file

```
require_once '/apps/myapp/controller/pagecontroller.php';
```

In other words: In order for the PageController class to be autoloaded, the class \\OCA\\MyApp\\Controller\\PageController needs to be stored in the /apps/myapp/controller/pagecontroller.php

Configuration

The config that allows the app to set *global*, *app*, and *user* settings can be injected from the ServerContainer. All values are saved as strings and must be cast to the correct value.

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;

use \OCA\MyApp\Service\AuthorService;


class Application extends App {

    public function __construct(array $urlParams=array()){
        parent::__construct('myapp', $urlParams);

        $container = $this->getContainer();

        /**
         * Controllers
         */
        $container->registerService('AuthorService', function($c) {
            return new AuthorService(
                $c->query('Config'),
                $c->query('AppName')
            );
        });

        $container->registerService('Config', function($c) {
            return $c->query('ServerContainer')->getConfig();
        });
    }
}
```

System Values

System values are saved in the config/config.php and allow the app to modify and read the global configuration:

```php
<?php
namespace OCA\MyApp\Service;

use \OCP\IConfig;


class AuthorService {

   private $config;
   private $appName;

   public function __construct(IConfig $config, $appName){
      $this->config = $config;
      $this->appName = $appName;
   }

   public function getSystemValue($key) {
      return $this->config->getSystemValue($key);
   }

   public function setSystemValue($key, $value) {
      $this->config->setSystemValue($key, $value);
   }

}
```

App Values

App values are saved in the database per application, and are useful for setting global application settings:

```php
<?php
namespace OCA\MyApp\Service;

use \OCP\IConfig;


class AuthorService {

   private $config;
   private $appName;

   public function __construct(IConfig $config, $appName){
      $this->config = $config;
      $this->appName = $appName;
   }

   public function getAppValue($key) {
      return $this->config->getAppValue($this->appName, $key);
   }

   public function setAppValue($key, $value) {
      $this->config->setAppValue($this->appName, $key, $value);
   }

}
```

User Values

User values are saved in the database per user and app and are good for saving user specific app settings:

```php
<?php
namespace OCA\MyApp\Service;

use \OCP\IConfig;


class AuthorService {

   private $config;
   private $appName;

   public function __construct(IConfig $config, $appName){
      $this->config = $config;
      $this->appName = $appName;
   }

   public function getUserValue($key, $userId) {
      return $this->config->getUserValue($userId, $this->appName, $key);
   }

   public function setUserValue($key, $userId, $value) {
      $this->config->setUserValue($userId, $this->appName, $key, $value);
   }

}
```

Routing

Routes map a URL and a method to a controller method. Routes are defined inside appinfo/routes.php by passing a configuration array to the registerRoutes method. An example route would look like this:

```php
<?php
namespace OCA\MyApp\AppInfo;

$application = new Application();
$application->registerRoutes($this, [
   'routes' => [
      ['name' => 'page#index', 'url' => '/', 'verb' => 'GET'],
   ]
]);
```

The route array contains the following parts:

- url: The URL that is matched after /index.php/apps/myapp ` name: The controller and the method to call; page#index is being mapped to PageController→index(), articles_api#drop_latest would be mapped to ArticlesApiController→dropLatest(). The controller that matches the page#index name would have to be registered in the following way inside appinfo/application.php:

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Controller\PageController;

class Application extends App {

   public function __construct(array $urlParams=[]){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('PageController', function($c) {
         return new PageController(
            $c->query('AppName'),
            $c->query('Request')
         );
      });
   }

}
```

- method (Optional, defaults to GET): The HTTP method that should be matched, (e.g., GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH) ` requirements (Optional): lets you match and extract URLs that have slashes in them (see Matching suburls) ` postfix (Optional): lets you define a route id postfix. Since each route name will be transformed to a route id (page#method → myapp.page.method) and the route id can only exist once you can use the postfix option to alter the route id creation by adding a string to the route id e.g.: 'name' ⇒ 'page#method', 'postfix' ⇒ 'test' will yield the route id myapp.page.methodtest. This makes it possible to add more than one route/url for a controller method ` defaults (Optional): If this setting is given, a default value will be assumed for each URL parameter which is not present. The default values are passed in as a key ⇒ value par array

Extracting Values From the URL

It is possible to extract values from the URL to allow for RESTful URL design. To extract value, you have to wrap it inside curly braces:

```php
<?php

// Request: GET /index.php/apps/myapp/authors/3

// appinfo/routes.php
['name' => 'author#show', 'url' => '/authors/\{id\}', 'verb' => 'GET'],

// controller/authorcontroller.php
class AuthorController {

   public function show($id) {
      // $id is '3'
   }

}
```

The identifier used inside the route is being passed into the controller method by reflecting the method parameters. To summarize, if you want to get the value of {id} in your method, you need to add $id to your method parameters.

Matching Sub-URLs

Sometimes you need to match more than one URL fragment. An example of this would be to match a request for all URLs that start with OPTIONS /index.php/apps/myapp/api. To do this, use the requirements parameter in your route, which is an array containing pairs of 'key' ⇒ 'regex':

```php
<?php

// Request: OPTIONS /index.php/apps/myapp/api/my/route

// appinfo/routes.php
[
  'name' => 'author_api#cors',
  'url' => '/api/{path}',
  'verb' => 'OPTIONS',
  'requirements' => ['path' => '.+']
],

// controller/authorapicontroller.php
class AuthorApiController {

   public function cors($path) {
      // $path will be 'my/route'
   }

}
```

Default Values for Sub-URL

Apart from matching requirements, a sub-URL may also have a default value. Say you want to support pagination (a page' parameter) for your `/posts sub-URL that displays posts entries list. You may set a default value for the page' parameter, that will be used if not already set in the URL. Use the `defaults parameter in your route which is an array containing pairs of 'urlparameter' ⇒ 'defaultvalue':

```
<?php

// Request: GET /index.php/app/myapp/post

// appinfo/routes.php
[
   'name'     => 'post#index',
   'url'      => '/post/{page}',
   'verb'     => 'GET',
   'defaults' => ['page' => 1] // this allows same url as /index.php/myapp/post/1
],

// controller/postcontroller.php
class PostController
{
   public function index($page = 1)
   {
      // $page will be 1
   }
}
```

Registering Resources

When dealing with resources, writing routes can become quite repetitive since most of the time routes for the following tasks are needed:

- Get all entries
- Get one entry by id
- Create an entry
- Update an entry
- Delete an entry

To prevent repetition, it’s possible to define resources. The following routes:

```php
<?php
namespace OCA\MyApp\AppInfo;

$application = new Application();
$application->registerRoutes($this, [
   'routes' => [
      ['name' => 'author#index', 'url' => '/authors', 'verb' => 'GET'],
      ['name' => 'author#show', 'url' => '/authors/{id}', 'verb' => 'GET'],
      ['name' => 'author#create', 'url' => '/authors', 'verb' => 'POST'],
      ['name' => 'author#update', 'url' => '/authors/{id}', 'verb' => 'PUT'],
      ['name' => 'author#destroy', 'url' => '/authors/{id}', 'verb' => 'DELETE'],
      // your other routes here
   ]
]);
```

can be abbreviated by using the resources key:

```php
<?php
namespace OCA\MyApp\AppInfo;

$application = new Application();
$application->registerRoutes($this, [
   'resources' => [
      'author' => ['url' => '/authors']
   ],
   'routes' => [
      // your other routes here
   ]
]);
```

Using the URLGenerator

Sometimes its useful to turn a route into a URL 1) to make the code independent from the URL design or to 2) generate an URL for an image in img/. For those use cases, the ServerContainer provides a service that can be used in your container:

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Controller\PageController;

class Application extends App {

    public function __construct(array $urlParams=[]){
        parent::__construct('myapp', $urlParams);

        $container = $this->getContainer();

        /**
         * Controllers
         */
        $container->registerService('PageController', function($c) {
            return new PageController(
                $c->query('AppName'),
                $c->query('Request'),

                // inject the URLGenerator into the page controller
                $c->query('ServerContainer')->getURLGenerator()
            );
        });
    }

}
```

Inside the PageController the URL generator can now be used to generate an URL for a redirect:

```php
<?php
namespace OCA\MyApp\Controller;

use \OCP\IRequest;
use \OCP\IURLGenerator;
use \OCP\AppFramework\Controller;
use \OCP\AppFramework\Http\RedirectResponse;

class PageController extends Controller {

   private $urlGenerator;

   public function __construct(
     $appName,
     IRequest $request,
     IURLGenerator $urlGenerator
   ) {
      parent::__construct($appName, $request);
      $this->urlGenerator = $urlGenerator;
   }

   /**
    * redirect to /apps/news/myapp/authors/3
    */
   public function redirect() {
      // route name: author_api#do_something
      // route url: /apps/news/myapp/authors/{id}

      // # needs to be replaced with a . due to limitations and prefixed
      // with your app id
      $route = 'myapp.author_api.do_something';
      $parameters = array('id' => 3);

      $url = $this->urlGenerator->linkToRoute($route, $parameters);

      return new RedirectResponse($url);
   }

}
```

URLGenerator is case-sensitive, so appName must match exactly the name you use in configuration <configuration>. If you use a camel-case name as *myCamelCaseApp,*

```php
<?php
$route = 'myCamelCaseApp.author_api.do_something';
```

Controllers

Controllers are used to connect routes <routes> with application logic. Think of them as callbacks that are executed once a request has come in. Controllers are defined inside the lib/Controller/ directory. To create a controller, extend the Controller class and create a method that should be executed to handle a request.

Here is an example of how to do so.

```
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

// define a new author controller
class AuthorController extends Controller {
   // define the method to execute upon the request
   public function index() {

   }
}
```

Connecting a Controller and a Route

To connect a controller and a route the controller has to be registered in the container like this:

```php
<?php
namespace OCA\MyApp\AppInfo;

use OCP\AppFramework\App;
use OCA\MyApp\Controller\AuthorApiController;

class Application extends application {

   public function __construct(array $urlParams=[]) {
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('AuthorApiController', function($c) {
         // register the controller in the container
         return new AuthorApiController(
            $c->query('AppName'),
            $c->query('Request')
         );
      });
   }
}
```

Every controller requires the application name and the request object to be passed to their parent constructor. This can be done as shown in the example code above.

The important part is not the class name, but rather the string which is passed in as the first parameter of the registerService method.

The other part is the route name. An example route name would look like this:

```
author_api#some_method
```

This name is processed in the following way:

1. Remove the underscore and uppercase the next character:

   ```
   authorApi#someMethod
   ```

2. Then split the name at the # and uppercase the first letter of the left part:

   ```
   AuthorApi
   someMethod
   ```

3. Then append Controller to the first part:

```
AuthorApiController
someMethod
```

4. Finally, retrieve the service listed under AuthorApiController from the container, look up the parameters of the someMethod method in the request, cast them if there are PHPDoc type annotations, and execute the someMethod method on the controller with those parameters.

Getting Request Parameters

Parameters can be passed in many ways, including:

- Extracting them from the URL using curly braces like {key} inside the URL (see routes)
- Appending them to the URL as a GET request (e.g. ?something=true)
- Setting the form's encoding type as application/x-www-form-urlencoded in a form request
- Setting the encoding type as application/json in a POST, PATCH, or PUT request

These parameters can be accessed by adding them to the controller method. For example:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

class PageController extends Controller {
   // this method will be executed with the id and name parameter taken
   // from the request
   public function doSomething($id, $name) {

   }
}
```

It is also possible to set default parameter values by using PHP default method values. This allows common values to be omitted. For example:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

class PageController extends Controller {
   /**
    * @param int $id
    */
   public function doSomething($id, $name='john', $job='author') {
      // GET ?id=3&job=killer
      // $id = 3
      // $name = 'john'
      // $job = 'killer'
   }
}
```

Casting Parameters

URL, GET and application/x-www-form-urlencoded have the problem that every parameter is a string, meaning that ?doMore=false would be passed in as the string 'false' which is not what one would expect. To cast these to the correct types, simply add a PHPDoc comment, in the form of @param type $name. Here's a comprehensive example showing all the options at once.

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

class PageController extends Controller {
   /**
    * @param int $id
    * @param bool $doMore
    * @param float $value
    */
   public function doSomething($id, $doMore, $value) {
      // GET /index.php/apps/myapp?id=3&doMore=false&value=3.5
      // => $id = 3
      //    $doMore = false
      //    $value = 3.5
   }
}
```

The following types will be cast:

- bool or boolean
- float
- int or integer

JSON Parameters

It is possible to pass JSON data using a POST, PUT or PATCH request. To do that the Content-Type header has to be set to application/json. The JSON will be parsed as an array. The first level keys will be used to pass in the arguments, e.g.:

```
POST /index.php/apps/myapp/authors
Content-Type: application/json
{
   "name": "test",
   "number": 3,
   "publisher": true,
   "customFields": {
      "mail": "test@example.com",
      "address": "Somewhere"
   }
}
```

```
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

class PageController extends Controller {
   public function create($name, $number, $publisher, $customFields) {
      // $name = 'test'
      // $number = 3
      // $publisher = true
      // $customFields = ["mail" => "test@example.com", "address" =>
"Somewhere"]
   }
}
```

Reading Headers, Files, Cookies and Environment Variables

Headers, files, cookies, and environment variables can be accessed directly from the request object:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\IRequest;

class PageController extends Controller {
   public function someMethod() {
      $type = $this->request->getHeader('Content-Type');  //
$_SERVER['HTTP_CONTENT_TYPE']
      $cookie = $this->request->getCookie('myCookie');    // $_COOKIES['myCookie']
      $file = $this->request->getUploadedFile('myfile');  // $_FILES['myfile']
      $env = $this->request->getEnv('SOME_VAR');          // $_ENV['SOME_VAR']
   }
}
```

Why should those values be accessed from the request object and not from the global array like $_FILES? Simple: because it's bad practice and will make testing harder.

Reading and Writing Session Variables

To set, get or modify session variables, the ISession object has to be injected into the controller. Then session variables can be accessed like this:

The session is closed automatically for writing, unless you add the @UseSession annotation!

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\ISession;
use OCP\IRequest;
use OCP\AppFramework\Controller;

class PageController extends Controller {

   private $session;

   public function __construct($AppName, IRequest $request, ISession $session)
{
      parent::__construct($AppName, $request);
      $this->session = $session;
   }

   /**
    * The following annotation is only needed for writing session values
    * @UseSession
    */
   public function writeASessionVariable() {
      // read a session variable
      $value = $this->session['value'];

      // write a session variable
      $this->session['value'] = 'new value';
   }
}
```

Setting Cookies

Cookies can be set or modified directly on the response class:

```php
<?php
namespace OCA\MyApp\Controller;

use DateTime;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\TemplateResponse;
use OCP\IRequest;

class BakeryController extends Controller {
   /**
    * Adds a cookie "foo" with value "bar" that expires after user closes the browser
    * Adds a cookie "bar" with value "foo" that expires 2015-01-01
    */
   public function addCookie() {
      $response = new TemplateResponse(...);
      $response->addCookie('foo', 'bar');
      $response->addCookie('bar', 'foo', new DateTime('2015-01-01 00:00'));
      return $response;
   }

   /**
    * Invalidates the cookie "foo"
    * Invalidates the cookie "bar" and "bazinga"
    */
   public function invalidateCookie() {
      $response = new TemplateResponse(...);
      $response->invalidateCookie('foo');
      $response->invalidateCookies(['bar', 'bazinga']);
      return $response;
   }
}
```

Responses

Similar to how every controller receives a request object, every controller method has to to return a Response. This can be in the form of a Response subclass or in the form of a value that can be handled by a registered responder.

JSON

Returning JSON is simple, just pass an array to a JSONResponse:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\JSONResponse;

class PageController extends Controller {
   public function returnJSON() {
      $params = ['test' => 'hi'];
      return new JSONResponse($params);
   }
}
```

Because returning JSON is such an common task, there's even a shorter way to do this:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;

class PageController extends Controller {
   public function returnJSON() {
      return ['test' => 'hi'];
   }
}
```

Why does this work? Because the dispatcher sees that the controller did not return a subclass of a Response and asks the controller to turn the value into a Response. That's where responders come in.

Responders

Responders are short functions that take a value and return a response. They are used to return different kinds of responses based on a format parameter which is supplied by the client. Think of an API that is able to return both XML and JSON depending on if you call the URL with:

```
?format=xml
```

or:

```
?format=json
```

The appropriate responder is being chosen by the following criteria:

- First the dispatcher checks the Request if there is a format parameter, e.g.:

```
?format=xml
```

or:

```
/index.php/apps/myapp/authors.{format}
```

- If there is none, take the Accept header, use the first mimetype and cut off application/. In the following example the format would be XML:

```
Accept: application/xml, application/json
```

- If there is no Accept header or the responder does not exist, format defaults to json.

By default there is only a responder for JSON but more can be added easily:

```
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\DataResponse;

class PageController extends Controller {

   public function returnHi() {
      // XMLResponse has to be implemented
      $this->registerResponder('xml', function($value) {
         if ($value instanceof DataResponse) {
            return new XMLResponse(
               $value->getData(),
               $value->getStatus(),
               $value->getHeaders()
            );
         } else {
            return new XMLResponse($value);
         }
      });

      return ['test' => 'hi'];
   }

}
```

The above example would only return XML if the format parameter was XML. If you want to return an XMLResponse regardless of the format parameter, extend the Response class and return a new instance of it from the controller method instead.

Because returning values works fine in case of a success but not in case of failure that requires a custom HTTP error code, you can always wrap the value in a DataResponse.

This works for both normal responses and error responses.

```
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\DataResponse;
use OCP\AppFramework\Http\Http;

class PageController extends Controller {

   public function returnHi() {
      try {
         return new DataResponse(calculate_hi());
      } catch (\Exception $ex) {
         return new DataResponse(['msg' => 'not found!'], Http
::STATUS_NOT_FOUND);
      }
   }

}
```

Templates

A template <templates> can be rendered by returning a TemplateResponse. A TemplateResponse takes the following parameters:

- appName: tells the template engine in which application the template should be located
- templateName: the name of the template inside the template/ folder without the .php extension
- parameters: optional array parameters that are available in the template through $_, e.g.:

  ```
  ['key' => 'something']
  ```

can be accessed through:

```
$_['key']
```

- renderAs: defaults to user, tells ownCloud if it should include it in the web interface, or in case blank is passed solely render the template

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\TemplateResponse;

class PageController extends Controller {
   public function index() {
      $templateName = 'main';  // will use templates/main.php
      $parameters = ['key' => 'hi'];
      return new TemplateResponse($this->appName, $templateName,
$parameters);
   }
}
```

Redirects

A redirect can be achieved by returning a RedirectResponse:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\RedirectResponse;

class PageController extends Controller {
   public function toGoogle() {
      return new RedirectResponse('https://google.com');
   }
}
```

Downloads

A file download can be triggered by returning a DownloadResponse:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\DownloadResponse;

class PageController extends Controller {
   public function downloadXMLFile() {
      $path = '/some/path/to/file.xml';
      $contentType = 'application/xml';

      return new DownloadResponse($path, $contentType);
   }
}
```

Creating Custom Responses

If no premade Response object fits the needed use case, its possible to extend the Response base class and create a custom one. The only thing that needs to be implemented is the render method which returns the result as string. Creating a custom XMLResponse class could look like this:

```php
<?php
namespace OCA\MyApp\Http;

use OCP\AppFramework\Http\Response;

class XMLResponse extends Response {

   private $xml;

   public function __construct(array $xml) {
      $this->addHeader('Content-Type', 'application/xml');
      $this->xml = $xml;
   }

   public function render() {
      $root = new SimpleXMLElement('<root/>');
      array_walk_recursive($this->xml, [$root, 'addChild']);
      return $xml->asXML();
   }
}
```

Streamed and Lazily Rendered Responses

By default all responses are rendered at once and sent as a string through middleware. In certain cases this is not a desirable behavior, for instance if you want to stream a file in order to save memory. To do that, use the OCP\\AppFramework\\Http\\StreamResponse class:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\StreamResponse;

class PageController extends Controller {

   public function downloadXMLFile() {
      return new StreamResponse('/some/path/to/file.xml');
   }
}
```

If you want to use a custom, lazily rendered response simply implement the interface OCP\\AppFramework\\Http\\ICallbackResponse for your response:

```php
<?php
namespace OCA\MyApp\Http;

use OCP\AppFramework\Http\Response;
use OCP\AppFramework\Http\ICallbackResponse;

class LazyResponse extends Response implements ICallbackResponse {
   public function callback(IOutput $output) {
      // custom code in here
   }
}
```

Because this code is rendered after several usually built in helpers, you need to take care of errors and proper HTTP caching by yourself.

Modifying the Content Security Policy

By default ownCloud disables all resources which are not served on the same domain, forbids cross domain requests and disables inline CSS and JavaScript by setting a Content Security Policy. However if an application relies on third party media or other features which are forbidden by the current policy the policy can be relaxed.

Double check your content and edge cases before you relax the policy! Also read the documentation provided by MDN

To relax the policy pass an instance of the Content Security Policy class to your response. The methods on the class can be chained. The following methods turn off security features by passing in true as the $isAllowed parameter:

- allowInlineScript (bool $isAllowed)
- allowInlineStyle (bool $isAllowed)
- allowEvalScript (bool $isAllowed)

The following methods whitelist domains by passing in a domain or * for any domain:

- addAllowedScriptDomain (string $domain)

- addAllowedStyleDomain (string $domain)
- addAllowedFontDomain (string $domain)
- addAllowedImageDomain (string $domain)
- addAllowedConnectDomain (string $domain)
- addAllowedMediaDomain (string $domain)
- addAllowedObjectDomain (string $domain)
- addAllowedFrameDomain (string $domain)
- addAllowedChildSrcDomain (string $domain)

The following policy for instance allows images, audio, and videos from other domains:

```
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http\TemplateResponse;
use OCP\AppFramework\Http\ContentSecurityPolicy;

class PageController extends Controller {
   public function index() {
      $response = new TemplateResponse('myapp', 'main');
      $csp = new ContentSecurityPolicy();
      $csp->addAllowedImageDomain('*');
         ->addAllowedMediaDomain('*');
      $response->setContentSecurityPolicy($csp);
   }
}
```

OCS

This is purely for compatibility reasons. If you are planning to offer an external API, go for a api instead.

In order to ease migration from OCS API routes to the application Framework, an additional controller and response have been added. To migrate your API you can use the OCP\\AppFramework\\OCSController base class and return your data in the form of an array in the following way:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\OCSController;

class ShareController extends OCSController {

   /**
    * @NoAdminRequired
    * @NoCSRFRequired
    * @PublicPage
    * @CORS
    */
   public function getShares() {
      return [
         'data' => [
            // actual data is in here
         ],
         // optional
         'statuscode' => 100,
         'status' => 'OK'
      ];
   }
}
```

The format parameter works out of the box, no intervention is required.

Handling Errors

Sometimes a request should fail, for instance if an author with id 1 is requested but does not exist. In that case use an appropriate HTTP error code to signal the client that an error occurred.

Each response subclass has access to the setStatus method which lets you set an HTTP status code. To return a JSONResponse signaling that the author with id 1 has not been found, use the following code:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\AppFramework\Controller;
use OCP\AppFramework\Http;
use OCP\AppFramework\Http\JSONResponse;

class AuthorController extends Controller {
   public function show($id) {
      try {
         // try to get author with $id

      } catch (NotFoundException $ex) {
         return new JSONResponse([], Http::STATUS_NOT_FOUND);
      }
   }
}
```

Authentication

By default every controller method enforces the maximum security, which is:

- Ensure that the user is admin
- Ensure that the user is logged in
- Check the CSRF token

Most of the time though it makes sense to also allow normal users to access the page and the PageController→index() method should not check the CSRF token because it has not yet been sent to the client and because of that can't work. To turn off checks the following *Annotations* can be added before the controller:

- @NoAdminRequired: Also users that are not admins can access the page
- @NoSubAdminRequired: Allow normal users access to the page
- @NoCSRFRequired: Don't check the CSRF token

Use this wisely since as you might create a security hole. To understand what it does see the Security Guidelines.

- @PublicPage: Everyone can access the page without having to log in

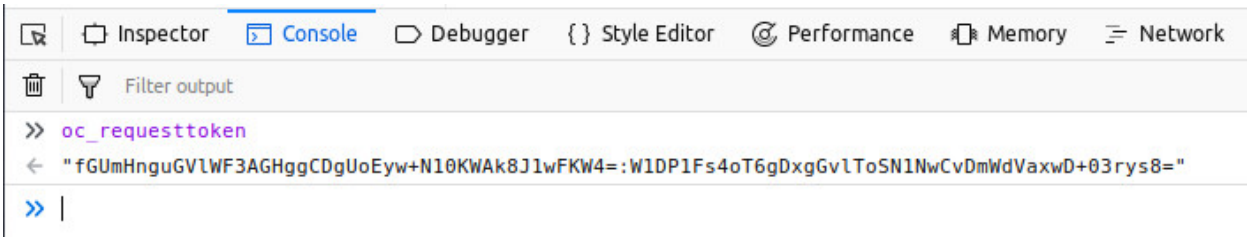
A controller method that turns off all checks would look like this:

```php
<?php
namespace OCA\MyApp\Controller;

use OCP\IRequest;
use OCP\AppFramework\Controller;

class PageController extends Controller {
   /**
    * @NoAdminRequired
    * @NoCSRFRequired
    * @PublicPage
    */
   public function freeForAll() {

   }
}
```

Using the CSRF Token in the DOM

The CSRF token is passed into the DOM automatically, and available in JavaScript via a global variable called oc_requesttoken. You can use this token in your Ajax requests via jQuery, as it is attached to your requests automatically. To debug it, open ownCloud in your browser, login, open the JavaScript console, and look at the value of oc_requesttoken.

Display the ownCloud CSRF token loaded into the DOM.



Templates

ownCloud provides its own templating system which is basically plain PHP with some additional functions and preset variables. All the parameters which have been passed from the controller <controllers> are available in an array called $_[], e.g.:

```
array('key' => 'something')
```

can be accessed through:

```
$_['key']
```

To prevent XSS the following PHP **functions for printing are forbidden: echo, print() and <?=**. Instead use the **p()** function for printing your values. Should you require unescaped printing, **double check for XSS** and use: :phpprint_unescaped.

Printing values is done by using the p() function, printing HTML is done by using print_unescaped().

templates/main.php

```
<?php foreach($_['entries'] as $entry){ ?>
  <p><?php p($entry); ?></p>
<?php
}
```

Including templates

Templates can also include other templates by using the $this→inc('templateName') method.

```
<?php print_unescaped($this->inc('sub.inc')); ?>
```

The parent variables will also be available in the included templates, but should you require it, you can also pass new variables to it by using the second optional parameter as array for $this→inc.

templates/sub.inc.php

```
<div>I am included, but I can still access the parents variables!</div>
<?php p($_['name']); ?>

<?php print_unescaped($this->inc('other_template', array('variable' => 'value')));
?>
```

Including CSS and JavaScript

To include CSS or JavaScript use the style and script functions:

```
<?php
script('myapp', 'script');  // add js/script.js
style('myapp', 'style');  // add css/style.css
```

Including images

To generate links to images use the image_path function:

```
<img src="<?php print_unescaped(image_path('myapp', 'app.png')); ?>" />
```

JavaScript

The JavaScript files reside in the js/ folder and should be included in the template:

```php
<?php
// add one file
script('myapp', 'script');  // adds js/script.js

// add multiple files in the same app
script('myapp', array('script', 'navigation'));  //  adds js/script.js js/navigation.js

// add vendor files (also allows the array syntax)
vendor_script('myapp', 'script');  //  adds vendor/script.js
```

If the script file is only needed when the file list is displayed, you should listen to the OCA\Files::loadAdditionalScripts event:

```php
<?php
$eventDispatcher = \OC::$server->getEventDispatcher();
$eventDispatcher->addListener('OCA\Files::loadAdditionalScripts', function() {
  script('myapp', 'script');  // adds js/script.js
  vendor_script('myapp', 'script');  //  adds vendor/script.js
});
```

Sending the CSRF Token

If any other JavaScript request library than jQuery is being used, the requests need to send the CSRF token as an HTTP header named requesttoken. The token is available in the global variable oc_requesttoken. For AngularJS the following lines would need to be added:

```
var app = angular.module('MyApp', []).config(['$httpProvider', function
($httpProvider) {
    $httpProvider.defaults.headers.common.requesttoken = oc_requesttoken;
}]);
```

Generating URLs

To send requests to ownCloud the base URL where ownCloud is currently running is needed. To get the base URL use:

```
var baseUrl = OC.generateUrl(`);
```

Full URLs can be generated by using:

```
var authorUrl = OC.generateUrl('/apps/myapp/authors/1');
```

Extending Core Parts

It is possible to extend components of the core web UI. The following examples should show how this is possible.

Extending the new Menu in the Files App

```
var myFileMenuPlugin = {
    attach: function (menu) {
        menu.addMenuEntry({
            id: 'abc',
            displayName: 'Menu display name',
            templateName: 'templateName.ext',
            iconClass: 'icon-filetype-text',
            fileType: 'file',
            actionHandler: function () {
                console.log('do something here');
            }
        });
    }
};
OC.Plugins.register('OCA.Files.NewFileMenu', myFileMenuPlugin);
```

This will register a new menu entry in the New menu of the files app. The method attach() is called once the menu is built. This usually happens right after the click on the button.

CSS

The CSS files reside in the css/ folder and should be included in the template:

```
<?php
// include one file
style('myapp', 'style');  // adds css/style.css

// include multiple files for the same app
style('myapp', ['style', 'navigation']);  // adds css/style.css, css/navigation.css

// include vendor file (also allows vendor syntax)
vendor_style('myapp', 'style');  // adds vendor/style.css
```

Web Components go into the component/ folder and can be imported like this:

```
<?php
// include one file
component('myapp', 'tabs');  // adds component/tabs.html

// include multiple files for the same app
component('myapp', ['tabs', 'forms']);  // adds component/tabs.html,
component/forms.html
```

Keep in mind that Web Components are still new and you might need to add polyfills using Polymer

Standard Layout

To use the commonly used layout consisting of sidebar navigation and content the app-navigation and app-content ids can be utilized:

```
<div id="app">
  <div id="app-navigation">Your navigation</div>
  <div id="app-content">
    <div id="app-content-wrapper">
      Your content in here
    </div>
  </div>
</div>
```

For built in mobile support your content has to be wrapped inside another div with the id app-content-wrapper.

Navigation

ownCloud provides a default CSS navigation layout. If list entries should have 16x16 px icons, the with-icon class can be added to the base ul. The maximum supported indention level is two; we do not recommend further indentations.

```
<div id="app-navigation">
  <ul class="with-icon">
    <li><a href="#">First level entry</a></li>
    <li>
      <a href="#">First level container</a>
      <ul>
        <li><a href="#">Second level entry</a></li>
        <li><a href="#">Second level entry</a></li>
      </ul>
    </li>
  </ul>
</div>
```

Folders

Folders are like normal entries and are only supported for the first level. In contrast to standard entries, the links which show the title of the folder need to have the icon-folder CSS class.

If the folder should be collapsible, the collapsible class and a button with the class collapse are needed. After adding the collapsible class the folder's child entries can be toggled by adding the open class to the list element:

```
<div id="app-navigation">
  <ul class="with-icon">
    <li><a href="#">First level entry</a></li>
    <li class="collapsible open">
      <button class="collapse"></button>
      <a href="#" class="icon-folder svg">Folder name</a>
      <ul>
        <li><a href="#">Folder contents</a></li>
        <li><a href="#">Folder contents</a></li>
      </ul>
    </li>
  </ul>
</div>
```

Drag and Drop

The class which should be applied to a first level element (li) that hosts or can host a second level is drag-and-drop. This will cause the hovered entry to slide down giving a visual hint that it can accept the dragged element. In the case of jQuery UI's droppable feature, the hoverClass option should be set to the drag-and-drop class.

```
<div id="app-navigation">
  <ul class="with-icon">
    <li><a href="#">First level entry</a></li>
    <li class="drag-and-drop">
      <a href="#" class="icon-folder svg">Folder name</a>
      <ul>
        <li><a href="#">Folder contents</a></li>
        <li><a href="#">Folder contents</a></li>
      </ul>
    </li>
  </ul>
</div>
```

Menus

To add actions that affect the current list element, you can add a menu for second and/or first level elements by adding the button and menu inside the corresponding li element and adding the with-menu CSS class:

```
<div id="app-navigation">
  <ul>
    <li class="with-counter with-menu">
      <a href="#">First level entry</a>

      <div class="app-navigation-entry-utils">
        <ul>
          <li class="app-navigation-entry-utils-counter">15</li>
          <li class="app-navigation-entry-utils-menu-button
svg"><button></button></li>
        </ul>
      </div>

      <div class="app-navigation-entry-menu open">
        <ul>
          <li><button class="icon-rename svg" title=
"rename"></button></li>
          <li><button class="icon-delete svg" title="delete"></button></li>
        </ul>
      </div>

    </li>
</div>
```

The div with the class app-navigation-entry-utils contains only the button (class: app-navigation-entry-utils-menu-button) to display the menu but in many cases, another entry is needed to display some sort of count (mails count, unread feed count, etc.). In that case, add the with-counter class to the list entry to adjust the correct padding and text-overflow of the entry's title.

The count should be limited to 999 and turn to 999+ if any higher number is given. If AngularJS is used the following filter can be used to get the correct behavior:

```
app.filter('counterFormatter', function () {
  'use strict';
  return function (count) {
    if (count > 999) {
      return '999+';
    }
    return count;
  };
});
```

Use it like this:

```
<li class="app-navigation-entry-utils-counter">{{ count | counterFormatter
}}</li>
```

The menu is hidden by default (display: none) and has to be triggered by adding the

open class to the app-navigation-entry-menu div. In the case of AngularJS the following small directive can be added to handle all the display and click logic out of the box:

```
app.run(function ($document, $rootScope) {
    'use strict';
    $document.click(function (event) {
        $rootScope.$broadcast('documentClicked', event);
    });
});

app.directive('appNavigationEntryUtils', function () {
    'use strict';
    return {
        restrict: 'C',
        function (scope, elm) {
            var menu = elm.siblings('.app-navigation-entry-menu');
            var button = $(elm)
                .find('.app-navigation-entry-utils-menu-button button');

            button.click(function () {
                menu.toggleClass('open');
            });

            scope.$on('documentClicked', function (scope, event) {
                if (event.target !== button[0]) {
                    menu.removeClass('open');
                }
            });
        }
    };
});
```

Editing

Often an edit option is needed for an entry. To add one for a given entry simply hide the title and add the following div inside the entry:

```
<div id="app-navigation">
  <ul class="with-icon">
    <li>
      <a href="#" class="hidden">First level entry</a>

      <div class="app-navigation-entry-edit">
        <form>
          <input type="text" value="First level entry" autofocus-on-insert>
          <input type="submit" value="" class="action icon-checkmark svg">
        </form>
      </div>

    </li>
  </ul>
</div>
```

If AngularJS is used you want to auto-focus the input box. This can be achieved by placing the show condition inside an ng-if on the app-navigation-entry-edit div and adding the following directive:

```
app.directive('autofocusOnInsert', function () {
  'use strict';
  return function (scope, elm) {
    elm.focus();
  };
});
```

ng-if is required because it removes/inserts the element into the DOM dynamically instead of just adding a display: none to it like ng-show and ng-hide.

Undo Entry

If you want to undo a performed action on a navigation entry such as deletion, you should show the undo directly in place of the entry and make it disappear after location change or seven seconds:

```
<div id="app-navigation">
  <ul class="with-icon">
    <li>
      <a href="#" class="hidden">First level entry</a>

      <div class="app-navigation-entry-deleted">
        <div class="app-navigation-entry-deleted-description">Deleted X</div>
        <button class="app-navigation-entry-deleted-button icon-history svg"
title="Undo"></button>
      </div>
    </li>
  </ul>
</div>
```

Settings Area

To create a settings area create a div with the id app-settings inside the app-navgiation div:

```
<div id="app">

  <div id="app-navigation">

    <!-- Your navigation here -->

    <div id="app-settings">
      <div id="app-settings-header">
        <button class="settings-button"
                data-apps-slide-toggle="#app-settings-content"
        ></button>
      </div>
      <div id="app-settings-content">
        <!-- Your settings in here -->
      </div>
    </div>
  </div>
</div>
```

The data attribute data-apps-slide-toggle slides up a target area using a jQuery selector and hides the area if the user clicks outside of it.

Icons

To use icons which are shipped in core, special classes to apply the background image are supplied. All of these classes use background-position: center and background-repeat: no-repeat.

Name	Image
icon-breadcrumb	
icon-loading	
icon-loading-dark	
icon-loading-small	
icon-add	+
icon-caret	
icon-caret-dark	▼
icon-checkmark	✔
icon-checkmark-white	
icon-clock	

Name	Image
icon-close	
icon-confirm	
icon-delete	
icon-download	
icon-history	
icon-info	
icon-lock	
icon-logout	
icon-mail	
icon-more	
icon-password	
icon-pause	
icon-pause-big	
icon-play	
icon-play-add	
icon-play-big	
icon-play-next	
icon-play-previous	
icon-public	
icon-rename	
icon-search	
icon-settings	
icon-share	
icon-shared	
icon-sound	
icon-sound-off	
icon-star	
icon-starred	
icon-toggle	
icon-triangle-e	
icon-triangle-n	
icon-triangle-s	
icon-upload	
icon-upload-white	
icon-user	

Name	Image
icon-view-close	
icon-view-next	
icon-view-pause	
icon-view-play	
icon-view-previous	
icon-calendar-dark	
icon-contacts-dark	
icon-file	
icon-files	
icon-folder	
icon-filetype-text	
icon-filetype-folder	
icon-home	
icon-link	
icon-music	
icon-picture	

Middleware

Middleware is logic that is run before and after each request and is modelled after Django's Middleware system. It offers the following hooks:

- **beforeController**: This is executed before a controller method is being executed. This allows you to plug additional checks or logic before that method, like for instance security checks
- **afterException**: This is being run when either the beforeController method or the controller method itself is throwing an exception. The middleware is asked in reverse order to handle the exception and to return a response. If the middleware can't handle the exception, it throws the exception again
- **afterController**: This is being run after a successful controller method call and allows the manipulation of a Response object. The middleware is run in reverse order
- **beforeOutput**: This is being run after the response object has been rendered and allows the manipulation of the outputted text. The middleware is run in reverse order

To generate your own middleware, simply inherit from the Middleware class and overwrite the methods that should be used.

```php
<?php

namespace OCA\MyApp\Middleware;

use \OCP\AppFramework\Middleware;

class CensorMiddleware extends Middleware {

   /**
    * this replaces "bad words" with "********" in the output
    */
   public function beforeOutput($controller, $methodName, $output){
      return str_replace('bad words', '********', $output);
   }

}
```

The middleware can be registered in the container and added using the registerMiddleware method:

```php
<?php

namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Middleware\CensorMiddleware;

class MyApp extends App {

    /**
     * Define your dependencies in here
     */
    public function __construct(array $urlParams=array()){
        parent::__construct('myapp', $urlParams);

        $container = $this->getContainer();

        /**
         * Middleware
         */
        $container->registerService('CensorMiddleware', function($c){
            return new CensorMiddleware();
        });

        // executed in the order that it is registered
        $container->registerMiddleware('CensorMiddleware');

    }
}
```

The order is important! The middleware that is registered first gets run first in the beforeController method. For all other hooks, the order is being reversed, meaning: if a middleware is registered first, it gets run last.

Parsing Annotations

Sometimes its useful to conditionally execute code before or after a controller method. This can be done by defining custom annotations. An example would be to add a custom authentication method or simply add an additional header to the response. To access the parsed annotations, inject the ControllerMethodReflector class:

```php
<?php

namespace OCA\MyApp\Middleware;

use \OCP\AppFramework\Middleware;
use \OCP\AppFramework\Utility\ControllerMethodReflector;
use \OCP\IRequest;

class HeaderMiddleware extends Middleware {

  private $reflector;

  public function __construct(ControllerMethodReflector $reflector) {
     $this->reflector = $reflector;
  }

  /**
   * Add custom header if @MyHeader is used
   */
  public function afterController($controller, $methodName, IResponse
$response){
     if($this->reflector->hasAnnotation('MyHeader')) {
        $response->addHeader('My-Header', 3);
     }
     return $response;
  }

}
```

Now adjust the container to inject the reflector:

```php
<?php

namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Middleware\HeaderMiddleware;

class MyApp extends App {

    /**
     * Define your dependencies in here
     */
    public function __construct(array $urlParams=array()){
        parent::__construct('myapp', $urlParams);

        $container = $this->getContainer();

        /**
         * Middleware
         */
        $container->registerService('HeaderMiddleware', function($c){
            return new HeaderMiddleware($c->query('ControllerMethodReflector'));
        });

        // executed in the order that it is registered
        $container->registerMiddleware('HeaderMiddleware');
    }

}
```

An annotation always starts with an uppercase letter.

Database Connectivity

Database Access

The basic way to run a database query is to use the database connection provided by OCP\\IDBConnection. Inside your database layer class you can now start running queries like:

```php
<?php
// db/authordao.php

namespace OCA\MyApp\Db;

use OCP\IDBConnection;

class AuthorDAO {

   private $db;

   public function __construct(IDBConnection $db) {
      $this->db = $db;
   }

   public function find($id) {
      $sql = 'SELECT * FROM `*PREFIX*myapp_authors` ' .
         'WHERE `id` = ?';
      $stmt = $this->db->prepare($sql);
      $stmt->bindParam(1, $id, \PDO::PARAM_INT);
      $stmt->execute();

      $row = $stmt->fetch();

      $stmt->closeCursor();
      return $row;
   }

}
```

Database programming guidelines

- Always use the Query Builder.
- Don't update more than 1 million rows within a transaction due to DB limitations.
- Don't add more than 999 conditions in a WHERE … IN … statement but chunk it into separate queries when using SQLite.
- When processing big tables, always do this in chunks, don't store the whole table in memory.
- Oracle compatibility specifics:
  - For Oracle, null and empty strings are the same thing. Special handling is required to catch these cases.
  - When reading values, make sure to convert nulls to empty strings when expected.
  - When using a condition based on empty strings, use is not null with Oracle instead.
  - Oracle can only compare the first 4000 bytes of a CLOB column.
  - Make sure to properly escape column names when using custom functions with createFunction. The escaping is usually done automatically by the query builder. Oracle is the most likely to complain about unquoted columns while other

databases will work fine.

    - Always add the table name when calling lastInsertId($tableName), as it is required by Oracle to return correct values.

- In general, don't specify a value for an autoincrement column. If you have to, keep in mind that Oracle's autoincrement trigger will get in the way on INSERT. As a result, you'll need a subsequent UPDATE to properly adjust the value.
- **Always** make sure there are unit tests for the database operations with queries to verify the result. This will help find out whether the database related code works on all databases and often times might reveal database quirks.
- Running unit tests with specific databases: make test-php TEST_PHP_SUITE=path/to/test/file.php TEST_DATABASE=$databasetype where "$databasetype" is one of "sqlite", "mysql", "mariadb", "pgsql", "oci" and "mysqlmb4".
- String concatenation should be done like this:
    - CONCAT(str1, str2, ... strN) for MYSQL.
    - str1 || str1 ... || strN SQLite/pgSQL/Oracle.
- Use IQueryBuilder::createPositionalParameter instead of IQueryBuilder::createNamedParameter when using like().

Mappers

The aforementioned example is the most basic way to write a simple database query but the more queries amass, the more code has to be written and the harder it will become to maintain it.

To generalize and simplify the problem, split code into resources and create an Entity and a Mapper class for it. The mapper class provides a way to run SQL queries and maps the result onto the related entities.

To create a mapper, inherit from the mapper baseclass and call the parent constructor with the following parameters:

- Database connection
- Table name
- **Optional**: Entity class name, defaults to \\OCA\\MyApp\\Db\\Author in the example below

```php
<?php
// db/authormapper.php

namespace OCA\MyApp\Db;

use OCP\IDBConnection;
use OCP\AppFramework\Db\Mapper;

class AuthorMapper extends Mapper {

    public function __construct(IDBConnection $db) {
        parent::__construct($db, 'myapp_authors');
    }


    /**
     * @throws \OCP\AppFramework\Db\DoesNotExistException if not found
     * @throws \OCP\AppFramework\Db\MultipleObjectsReturnedException if more
than one result
     */
    public function find($id) {
        $sql = 'SELECT * FROM `*PREFIX*myapp_authors` ' .
            'WHERE `id` = ?';
        return $this->findEntity($sql, [$id]);
    }


    public function findAll($limit=null, $offset=null) {
        $sql = 'SELECT * FROM `*PREFIX*myapp_authors`';
        return $this->findEntities($sql, $limit, $offset);
    }


    public function authorNameCount($name) {
        $sql = 'SELECT COUNT(*) AS `count` FROM `*PREFIX*myapp_authors` ' .
            'WHERE `name` = ?';
        $stmt = $this->execute($sql, [$name]);

        $row = $stmt->fetch();
        $stmt->closeCursor();
        return $row['count'];
    }

}
```

The cursor is closed automatically for all **INSERT**, **DELETE**, **UPDATE** queries and when calling the methods **findOneQuery**, **findEntities**, **findEntity**, **delete**, **insert** and **update**. For custom calls using execute you should always close the cursor after you are done with the fetching to prevent database lock problems on SqLite

Every mapper also implements default methods for deleting and updating an entity based on its id:

```
$authorMapper->delete($entity);
```

or:

```
$authorMapper->update($entity);
```

Entities

Entities are data objects that carry all the table's information for one row. Every Entity has an id field by default that is set to the integer type. Table rows are mapped from lower case and underscore separated names to pascal case attributes:

- **Table column name**: phone_number
- **Property name**: phoneNumber

```
<?php
// db/author.php
namespace OCA\MyApp\Db;

use OCP\AppFramework\Db\Entity;

class Author extends Entity {

   protected $stars;
   protected $name;
   protected $phoneNumber;

   public function __construct() {
      // add types in constructor
      $this->addType('stars', 'integer');
   }
}
```

Types

The following properties should be annotated by types, to not only assure that the types are converted correctly for storing them in the database (e.g., PHP casts false to the empty string which fails on PostgreSQL) but also for casting them when they are retrieved from the database.

The following types can be added for a field:

- integer
- float
- boolean

Accessing attributes

Since all attributes should be protected, getters and setters are automatically generated for you:

```
<?php
// db/author.php
namespace OCA\MyApp\Db;

use OCP\AppFramework\Db\Entity;

class Author extends Entity {
   protected $stars;
   protected $name;
   protected $phoneNumber;
}

$author = new Author();
$author->setId(3);
$author->getPhoneNumber()  // null
```

Custom Attribute to Database Column Mapping

By default each attribute will be mapped to a database column by a certain convention, e.g. phoneNumber will be mapped to the column phone_number and vice versa. Sometimes it is needed though to map attributes to different columns because of backwards compatibility. To define a custom mapping, simply override the columnToProperty and propertyToColumn methods of the entity in question:

```php
<?php
// db/author.php
namespace OCA\MyApp\Db;

use OCP\AppFramework\Db\Entity;

class Author extends Entity {
   protected $stars;
   protected $name;
   protected $phoneNumber;

   // map attribute phoneNumber to the database column phonenumber
   public function columnToProperty($column) {
      if ($column === 'phonenumber') {
         return 'phoneNumber';
      } else {
         return parent::columnToProperty($column);
      }
   }

   public function propertyToColumn($property) {
      if ($column === 'phoneNumber') {
         return 'phonenumber';
      } else {
         return parent::propertyToColumn($property);
      }
   }

}
```

Slugs

Slugs are used to identify resources in the URL by a string rather than integer id. Since the URL allows only certain values, the entity baseclass provides a slugify method for it:

```php
<?php
$author = new Author();
$author->setName('Some*thing');
$author->slugify('name');  // Some-thing
```

Database Migrations

ownCloud uses migration steps to perform changes between releases. In most cases, these changes relate to the core database schema. However, other types of changes may be required. Therefore we support three kinds of migration steps, these are:

- **Simple:** run general migration steps. These are quite similar to the migration repair steps.
- **SQL:** create a list of executable SQL commands.

- **Schema:** migration via schema migration operations.

Starting with ownCloud 10, this is the preferred way to perform any kind of migrations and is enabled by default within core. Any app which wants to use this mechanism has to enable it in appinfo/info.xml, by adding the following:

```
<use-migrations>true</use-migrations>
```

Please Be Aware: if migrations are enabled then appinfo/database.xml is ignored. From this point onwards, when an app is installed or upgraded, all outstanding migrations are executed. Below is a migration code sample for creating an application's core table.

```php
<?php

namespace OCA\MyApp\Migrations;

use OCP\Migration\ISchemaMigration;
use Doctrine\DBAL\Schema\Schema;

/*
 - Create initial tables for the app
 */

class Version20171106150538 implements ISchemaMigration {

   /** @var  string */
   private $prefix;

   /**
    - @param Schema $schema
    - @param [] $options
    */
   public function changeSchema(Schema $schema, array $options) {
      $this->prefix = $options['tablePrefix'];

      if (!$schema->hasTable("{$this->prefix}mytable")) {
         $table = $schema->createTable("{$this->prefix}mytable");
         $table->addColumn('id', 'integer', [
            'autoincrement' => true,
            'unsigned' => true,
            'notnull' => true,
            'length' => 11,
         ]);
         $table->addColumn('stringfield', 'string', [
            'length' => 255,
            'notnull' => false,
         ]);
         $table->addColumn('intfield', 'integer', [
            'unsigned' => true,
            'notnull' => true,
            'default' => 1,
         ]);
         $table->setPrimaryKey(['id']);
         $table->addUniqueIndex(['stringfield'], 'mytable_index');
      }
   }
}
```

You can see examples of how to create the three migration types in the next section.

It is still necessary to increment the application's version number to trigger the execution of migrations.

How to Create a Migration

1. Enable migrations by adding the XML tag to appinfo/info.xml

```
<use-migrations>true</use-migrations>
```

1. Create a migration step

```
./occ migrations:generate app-name {simple, SQL, schema}
```

A Simple Migration Step

The simple migration step skeleton looks like this:

```
<?php
namespace OCA\testing\Migrations;

use OCP\Migration\ISimpleMigration;
use OCP\Migration\IOutput;

/**
 * Auto-generated migration step: Please modify to your needs!
 */
class Version20170213125339 implements ISimpleMigration {
    /**
     * @param IOutput $out
     */
    public function run(IOutput $out) {
        // auto-generated - please modify it to your needs
    }
}
```

A SQL Migration Step

A SQL migration step skeleton looks like this:

```php
<?php
namespace OCA\testing\Migrations;

use OCP\IDBConnection;
use OCP\Migration\ISqlMigration;

/**
 * Auto-generated migration step: Please modify to your needs!
 */
class Version20170213125430 implements ISqlMigration {

    /**
     * @param IDBConnection $connection
     * @return array of sql statements
     */
    public function sql(IDBConnection $connection) {
        // auto-generated - please modify it to your needs
    }
}
```

Within the sql() method you can generate any number of SQL commands. The generated commands will be returned as an array, and the statements will be executed afterward.

Please do not execute any generated SQL statements directly on the database.

The parameter $connection can be used to retrieve a database platform object or to test if tables exist. In order to create cross-compatible SQL code, please use the platform object or generate SQL commands for each supported database system.

A Schema Migration Step

A schema migration step skeleton looks like this:

```php
<?php
namespace OCA\testing\Migrations;

use Doctrine\DBAL\Schema\Schema;
use OCP\Migration\ISchemaMigration;

/**
 * Auto-generated migration step: Please modify to your needs!
 */
class Version20170213125427 implements ISchemaMigration {
    public function changeSchema(Schema $schema, array $options) {
        // auto-generated - please modify it to your needs
    }
}
```

Within the changeSchema() method, you can use the https://www.doctrine-project.org/api/dbal/2.9/Doctrine/DBAL/Schema/Schema.htmlClass Schema] to manipulate the

existing database schema. This is the preferred way to manipulate the schema.

1. Test your migration step

```
./occ migrations:execute dav 20161130090952
```

Because all migration steps will be executed upon installation, there is no explicit need for unit tests.

1. Deploy the migration(s)

To trigger the migrations, the app version has to be increased. Doing so applies all steps which have not yet been executed.

How to Update the Database Schema

ownCloud uses a database abstraction layer on top of PDO, depending on its availability on the server. The database schema is contained in appinfo/database.xml, and uses MDB2's XML scheme notation. The placeholders **dbprefix** (**PREFIX** in your SQL) and **dbname** can be used for the configured database table prefix and database name.

An example database XML file would look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<database>
 <name>*dbname*</name>
 <create>true</create>
 <overwrite>false</overwrite>
 <charset>utf8</charset>
 <table>
  <name>*dbprefix*yourapp_items</name>
  <declaration>
   <field>
    <name>id</name>
    <type>integer</type>
    <default>0</default>
    <notnull>true</notnull>
      <autoincrement>1</autoincrement>
    <length>4</length>
   </field>
   <field>
    <name>user</name>
    <type>text</type>
    <notnull>true</notnull>
    <length>64</length>
   </field>
   <field>
    <name>name</name>
    <type>text</type>
    <notnull>true</notnull>
    <length>100</length>
   </field>
   <field>
    <name>path</name>
    <type>clob</type>
    <notnull>true</notnull>
   </field>
  </declaration>
</table>
</database>
```

To update the tables used by the app: adjust the database.xml file to reflect the changes which you want to make. Then, increment the app version number in appinfo/info.xml to trigger an update.

Background Jobs

ownCloud supports background job functionality (otherwise known as Cron jobs). To create them requires two steps to be completed:

- Create a job class
- Register the class with ownCloud

Create a Job Class

The first step is to create a job class, which will provide the job functionality. For this example, we will call it: lib/Cron/SomeTask.php. The class only needs to define one, static, method called run. In this example, we're retrieving a service from the container, and in turn calling its run method.

```
<?php

namespace OCA\MyApp\Cron;

use \OCA\MyApp\AppInfo\Application;

class SomeTask extends OC\BackgroundJob\TimedJob {

   protected function run($argument) {
      (new Application())
         ->getContainer()
         ->query('SomeService')
         ->run();
   }
}
```

Try to keep the method as small as possible, because its hard to test static methods.

Register the Class with ownCloud

Next, you need to register the job as a background job. This is done in appinfo/info.xml by adding a job element, containing the name of the job class, to the background-jobs element. The example below shows how to add the SomeTask class, which we just created, as a background job.:

```
<background-jobs>
   <job>\OCA\MyApp\Cron\SomeTask</job>
</background-jobs>
```

Testing

To test the job classes, you can run Cron manually, as in the example below:

```
sudo -u www-data php cron.php
```

After doing so, you will need to reset the job to allow it to be run, manually, again. To do this, go to the database and run the following SQL query:

```
UPDATE oc_jobs SET last_run=0,last_checked=0,reserved_at=0;
```

Is The Cron Service Running?

Finally, don't forget to add the ownCloud Cron process in the web server's crontab. To do this, first open the web server's crontab for editing by running:

In this example, www-data is the web server user:

```
sudo crontab -u www-data -e
```

Then, add the ownCloud Cron process to the crontab, for example:

```
*/15  *  *  *  * php -f /var/www/owncloud/cron.php
```

Logging

The logger can be injected from the ServerContainer:

```
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Service\AuthorService;

class Application extends App {

   public function __construct(array $urlParams=[]){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('AuthorService', function($c) {
         return new AuthorService(
            $c->query('Logger'),
            $c->query('AppName')
         );
      });

      $container->registerService('Logger', function($c) {
         return $c->query('ServerContainer')->getLogger();
      });
   }
}
```

and then be used in the following way:

```php
<?php
namespace OCA\MyApp\Service;

use \OCP\ILogger;

class AuthorService {

   private $logger;
   private $appName;

   public function __construct(ILogger $logger, $appName){
      $this->logger = $logger;
      $this->appName = $appName;
   }

   public function log($message) {
      $this->logger->error($message, ['app' => $this->appName]);
   }

}
```

The following methods are available:

- emergency
- alert
- critical
- error
- warning
- notice
- info
- debug

Which Logging Level Should You Use?

When considering which logging level to use, please refer to this guide from IG:

DEBUG
: Information that is useful during development. Usually very chatty, and will not show in production.

INFO
: Information you will need to debug production issues.

WARN (warning)
: Someone in the team will have to investigate what happened, but it can wait until tomorrow.

ERROR
: Oh-oh, call the fireman! This needs to be investigated **now!**

Further Reading

- ownCloud Logging Configuration documentation.

Testing

All PHP classes can be tested with PHPUnit, JavaScript can be tested by using Karma.

PHP

The PHP tests go into the tests/ directory. Unfortunately the classloader in core requires a running server (as in a fully configured and running setup up with a database connection). This is, unfortunately, too complicated and slow so a separate classloader has to be provided.

When writing your own tests, please ensure that PHPUnit bootstraps from tests/bootstrap.php, to set up various environment variables and autoloader registration correctly. Without this, you will see errors as the ownCloud autoloader security policy prevents access to the tests/ subdirectory. This can be configured in your phpunit.xml file as follows:

```
<phpunit bootstrap="../../tests/bootstrap.php">
```

PHP classes should be tested by accessing them from the container to ensure that the container is wired up properly. Services that should be mocked can be replaced directly in the container. A test for the AuthorStorage class in filesystem:

```
<?php
namespace OCA\MyApp\Storage;

class AuthorStorage {

   private $storage;

   public function __construct($storage){
      $this->storage = $storage;
   }

   public function getContent($id) {
      // check if file exists and write to it if possible
      try {
         $file = $this->storage->getById($id);
         if($file instanceof \OCP\Files\File) {
            return $file->getContent();
         } else {
            throw new StorageException('Can not read from folder');
         }
      } catch(\OCP\Files\NotFoundException $e) {
         throw new StorageException('File does not exist');
      }
   }
}
```

would look like this:

```
<?php
// tests/Storage/AuthorStorageTest.php
namespace OCA\MyApp\Tests\Storage;

class AuthorStorageTest extends \Test\TestCase {

   private $container;
   private $storage;

   protected function setUp() {
      parent::setUp();

      $app = new \OCA\MyApp\AppInfo\Application();
      $this->container = $app->getContainer();
      $this->storage = $storage = $this->getMockBuilder('\OCP\Files\Folder')
         ->disableOriginalConstructor()
         ->getMock();

      $this->container->registerService('RootStorage', function($c) use ($storage)
{
         return $storage;
      });
   }

   /**
    * @expectedException \OCA\MyApp\Storage\StorageException
    */
   public function testFileNotFound() {
      $this->storage->expects($this->once())
         ->method('get')
         ->with($this->equalTo(3))
         ->will($this->throwException(new \OCP\Files\NotFoundException()));

      $this->container['AuthorStorage']->getContent(3);
   }

}
```

Make sure to extend the \Test\TestCase class with your test and always call the parent methods, when overwriting setUp(), setUpBeforeClass(), tearDown() or tearDownAfterClass() method from the TestCase. These methods set up important stuff and clean up the system after the test, so the next test can run without side effects, like remaining files and entries in the file cache, etc.

The DI Container

The App Framework assembles the application by using a container based on the software pattern Dependency Injection. This makes the code easier to test and thus easier to maintain.

If you are unfamiliar with this pattern, watch the following videos:

- Dependency Injection and the art of Services and Containers Tutorial
- Google Clean Code Talks

Dependency Injection

Dependency Injection sounds pretty complicated but it just means: Don't put new dependencies in your constructor or methods but pass them in. So this:

```
<?php

// without dependency injection
class AuthorMapper {

  private $db;

  public function __construct() {
    $this->db = new Db();
  }

}
```

would turn into this by using Dependency Injection:

```
<?php

// with dependency injection
class AuthorMapper {

  private $db;

  public function __construct($db) {
    $this->db = $db;
  }

}
```

Using a Container

Passing dependencies into the constructor rather than instantiating them in the constructor has the following drawback: Every line in the source code where new AuthorMapper is being used has to be changed, once a new constructor argument is being added to it.

The solution for this particular problem is to limit the new AuthorMapper to one file, the container. The container contains all the factories for creating these objects and is configured in lib/AppInfo/Application.php.

To add the app's classes simply open the lib/AppInfo/Application.php and use the registerService method on the container object:

```php
<?php

namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;

use \OCA\MyApp\Controller\AuthorController;
use \OCA\MyApp\Service\AuthorService;
use \OCA\MyApp\Db\AuthorMapper;

class Application extends App {


 /**
  * Define your dependencies in here
  */
 public function __construct(array $urlParams=[]){
  parent::__construct('myapp', $urlParams);

  $container = $this->getContainer();

  /**
   * Controllers
   */
  $container->registerService('AuthorController', function($c){
   return new AuthorController(
    $c->query('AppName'),
    $c->query('Request'),
    $c->query('AuthorService')
   );
  });

  /**
   * Services
   */
  $container->registerService('AuthorService', function($c){
   return new AuthorService(
    $c->query('AuthorMapper')
   );
  });

  /**
   * Mappers
   */
  $container->registerService('AuthorMapper', function($c){
   return new AuthorMapper(
    $c->query('ServerContainer')->getDb()
   );
  });
 }
```

```
}
```

How the Container Works

The container works in the following way:

- A request comes in and is matched against a route (for the AuthorController in this case)
- The matched route queries AuthorController service from the container:

```
return new AuthorController(
  $c->query('AppName'),
  $c->query('Request'),
  $c->query('AuthorService')
);
```

- The AppName is queried and returned from the baseclass
- The Request is queried and returned from the server container
- AuthorService is queried:

```
$container->registerService('AuthorService', function($c){
  return new AuthorService(
    $c->query('AuthorMapper')
  );
});
```

- AuthorMapper is queried:

```
$container->registerService('AuthorMappers', function($c){
  return new AuthorService(
    $c->query('ServerContainer')->getDb()
  );
});
```

- The database connection is returned from the server container
- Now AuthorMapper has all of its dependencies and the object is returned
- AuthorService gets the AuthorMapper and returns the object
- AuthorController gets the AuthorService and finally the controller can be instantiated and the object is returned

So basically the container is used as a giant factory to build all the classes that are needed for the application. Because it centralizes all the creation of objects (the new Class() lines), it is very easy to add new constructor parameters without breaking existing code: only the __construct method and the container line where the new is being called need to be changed.

Use Automatic Dependency Assembly (Recommended)

Since ownCloud 8 it is possible to omit the lib/AppInfo/Application.php and use automatic dependency assembly instead.

How Does Automatic Assembly Work

Automatic assembly creates new instances of classes just by looking at the class name and its constructor parameters. For each constructor parameter the type or the variable name is used to query the container, e.g.:

- SomeType $type will use $container→query('SomeType')
- $variable will use $container→query('variable')

If all constructor parameters are resolved, the class will be created, saved as a service and returned. So basically the following is now possible:

```
<?php
namespace OCA\MyApp;

class MyTestClass {}

class MyTestClass2 {
   public $class;
   public $appName;

   public function __construct(MyTestClass $class, $AppName) {
      $this->class = $class;
      $this->appName = $AppName;
   }
}

$app = new \OCP\AppFramework\App('myapp');

$class2 = $app->getContainer()->query('OCA\MyApp\MyTestClass2');

$class2 instanceof MyTestClass2;  // true
$class2->class instanceof MyTestClass;  // true
$class2->appName === 'myapp';  // true
$class2 === $app->getContainer()->query('OCA\MyApp\MyTestClass2');  // true
```

$AppName is resolved because the container registered a parameter under the key AppName' which will return the app id. The lookup is case sensitive so while `$AppName will work correctly, using $appName as a constructor parameter will fail.

How Does it Affect the Request Lifecycle

- A request comes in
- All apps' routes.php files are loaded
  - If a routes.php file returns an array, and an appname/lib/AppInfo/Application.php exists, include it, create a new instance of \\OCA\\AppName\\AppInfo\\Application.php and register the routes on it. That way a container can be used while still benefitting from the new routes behavior
  - If a routes.php file returns an array, but there is no appname/lib/AppInfo/Application.php, create a new \OCP\AppFramework\App instance with the app id and register the routes on it
- A request is matched for the route, e.g. with the name page#index

- The appropriate container is being queried for the entry PageController (to keep backwards compability)
- If the entry does not exist, the container is queried for OCA\AppName\Controller\PageController and if no entry exists, the container tries to create the class by using reflection on its constructor parameters

How Does This Affect Controllers

The only thing that needs to be done to add a route and a controller method is now:

myapp/appinfo/routes.php

```
<?php
return ['routes' => [
   ['name' => 'page#index', 'url' => '/', 'verb' => 'GET'],
]];
```

myapp/appinfo/lib/Controller/PageController.php

```
<?php
namespace OCA\MyApp\Controller;

class PageController {
   public function __construct($AppName, \OCP\IRequest $request) {
      parent::__construct($AppName, $request);
   }

   public function index() {
      // your code here
   }
}
```

There is no need to wire up anything in lib/AppInfo/Application.php. Everything will be done automatically.

How to Deal with Interface and Primitive Type Parameters

Interfaces and primitive types can not be instantiated, so the container can not automatically assemble them. The actual implementation needs to be wired up in the container:

```php
<?php

namespace OCA\MyApp\AppInfo;

class Application extends \OCP\AppFramework\App {

   /**
    * Define your dependencies in here
    */
   public function __construct(array $urlParams=[]){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      // AuthorMapper requires a location as string called $TableName
      $container->registerParameter('TableName', 'my_app_table');

      // the interface is called IAuthorMapper and AuthorMapper implements it
      $container->registerService('OCA\MyApp\Db\IAuthorMapper', function ($c) {
         return $c->query('OCA\MyApp\Db\AuthorMapper');
      });
   }

}
```

Predefined Core Services

The following parameter names and type hints can be used to inject core services instead of using $container→getServer()→getServiceX()

Parameters:

- AppName: The app id
- WebRoot: The path to the ownCloud installation
- UserId: The id of the current user

Types:

- OCP\\IAppConfig
- OCP\\IAppManager
- OCP\\IAvatarManager
- OCP\\Activity\\IManager
- OCP\\ICache
- OCP\\ICacheFactory
- OCP\\IConfig
- OCP\\AppFramework\\Utility\\IControllerMethodReflector
- OCP\\Contacts\\IManager
- OCP\\IDateTimeZone
- OCP\\IDb

- OCP\\IDBConnection
- OCP\\Diagnostics\\IEventLogger
- OCP\\Diagnostics\\IQueryLogger
- OCP\\Files\\Config\\IMountProviderCollection
- OCP\\Files\\IRootFolder
- OCP\\IGroupManager
- OCP\\IL10N
- OCP\\ILogger
- OCP\\BackgroundJob\\IJobList
- OCP\\INavigationManager
- OCP\\IPreview
- OCP\\IRequest
- OCP\\AppFramework\\Utility\\ITimeFactory
- OCP\\ITagManager
- OCP\\ITempManager
- OCP\\Route\\IRouter
- OCP\\ISearch
- OCP\\ISearch
- OCP\\Security\\ICrypto
- OCP\\Security\\IHasher
- OCP\\Security\\ISecureRandom
- OCP\\IURLGenerator
- OCP\\IUserManager
- OCP\\IUserSession

How to Enable It

To make use of this new feature, the following things have to be done:

- appinfo/info.xml requires to provide another field called namespace where the namespace of the app is defined. The required namespace is the one which comes after the top level namespace OCA\\, e.g.: for OCA\\MyBeautifulApp\\Some\\OtherClass the needed namespace would be MyBeautifulApp and would be added to the info.xml in the following way:

```
<?xml version="1.0"?>
<info>
  <namespace>MyBeautifulApp</namespace>
  <!-- other options here ... -->
</info>
```

- appinfo/routes.php: Instead of creating a new Application class instance, simply return the routes array like:

```
<?php
return ['routes' => [
   ['name' => 'page#index', 'url' => '/', 'verb' => 'GET'],
]];
```

A namespace tag is required because you can not deduce the namespace from the app id

Which Classes Should Be Added

In general all of the app's controllers need to be registered inside the container. Then the following question is: What goes into the constructor of the controller? Pass everything into the controller constructor that matches one of the following criteria:

- It does I/O (database, write/read to files)
- It is a global (e.g. $_POST, etc. This is in the request class by the way)
- The output does not depend on the input variables (also called impure function), e.g. time, random number generator
- It is a service, basically it would make sense to swap it out for a different object

What not to inject:

- It is pure data and has methods that only act upon it (arrays, data objects)
- It is a pure function

Filesystem

Because users can choose their storage backend, the filesystem should be accessed by using the appropriate filesystem classes. Filesystem classes can be injected from the ServerContainer by calling the method getRootFolder(), getUserFolder() or getAppFolder():

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Storage\AuthorStorage;

class Application extends App {

   public function __construct(array $urlParams=[]){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Storage Layer
       */
      $container->registerService('AuthorStorage', function($c) {
         return new AuthorStorage($c->query('RootStorage'));
      });

      $container->registerService('RootStorage', function($c) {
         return $c->query('ServerContainer')->getRootFolder();
      });

   }
}
```

Writing to a File

All methods return a Folder object on which files and folders can be accessed, or filesystem operations can be performed relatively to their root. For instance for writing to owncloud/data/myfile.txt you should get the root folder and use:

```php
<?php
namespace OCA\MyApp\Storage;

class AuthorStorage {

   private $storage;

   public function __construct($storage){
      $this->storage = $storage;
   }

   public function writeTxt($content) {
      // check if file exists and write to it if possible
      try {
         try {
            $file = $this->storage->get('/myfile.txt');
         } catch(\OCP\Files\NotFoundException $e) {
            $file = $this->storage->newFile('/myfile.txt');
         }

         // the id can be accessed by $file->getId();
         $file->putContent($content);

      } catch(\OCP\Files\NotPermittedException $e) {
         // you have to create this exception by yourself ;)
         throw new StorageException('Cant write to file');
      }
   }
}
```

Reading from a File

Files and folders can also be accessed by id, by calling the getById method on the folder.

```php
<?php
namespace OCA\MyApp\Storage;

class AuthorStorage {

   private $storage;

   public function __construct($storage){
      $this->storage = $storage;
   }

   public function getContent($id) {
      // check if file exists and write to it if possible
      try {
         $file = $this->storage->getById($id);
         if($file instanceof \OCP\Files\File) {
            return $file->getContent();
         } else {
            throw new StorageException('Can not read from folder');
         }
      } catch(\OCP\Files\NotFoundException $e) {
         throw new StorageException('File does not exist');
      }
   }
}
```

How to Get the Storage Owner Using a File Id

A storage's owner can be retrieved using a file id, as in the following example.

```php
<?php

$mountCache = \OC::$server->getMountProviderCollection()->getMountCache();

$mounts = $mountCache->getMountsForFileId($fileId);
$userWithAccessToFile = array_map(function(ICachedMountInfo $mount) {
   return $mount->getUser();
}, $mounts);

$mounts = $mountCache->getMountsForFileId($fileId);
if (count($mounts) > 0) {
   $node = $mounts[0]->getMountPointNode();
   $owner = $node->getOwner();
}
```

RESTful API

Offering a RESTful API is not different from creating a route <routes> and controllers <controllers> for the web interface. It is recommended though to inherit from

ApiController and add **@CORS** annotations to the methods so that web applications will also be able to access the API.

```
<?php
namespace OCA\MyApp\Controller;

use \OCP\AppFramework\ApiController;
use \OCP\IRequest;

class AuthorApiController extends ApiController {

   public function __construct($appName, IRequest $request) {
      parent::__construct($appName, $request);
   }

   /**
    * @CORS
    */
   public function index() {

   }

}
```

CORS also needs a separate URL for the preflighted **OPTIONS** request that can easily be added by adding the following route:

```
<?php
// appinfo/routes.php
array(
   'name' => 'author_api#preflighted_cors',
   'url' => '/api/1.0/{path}',
   'verb' => 'OPTIONS',
   'requirements' => array('path' => '.+')
)
```

Keep in mind that multiple apps will likely depend on the API interface once it is published and they will move at different speeds to react to changes implemented in the API. Therefore it is recommended to version the API in the URL to not break existing apps when backwards incompatible changes are introduced:

```
/index.php/apps/myapp/api/1.0/resource
```

Modifying the CORS headers

By default the following values will be used for the preflighted OPTIONS request:

- **Access-Control-Allow-Methods**: `PUT, POST, GET, DELETE, PATCH'
- **Access-Control-Allow-Headers**: `Authorization, Content-Type, Accept'

- **Access-Control-Max-Age**: 1728000

To add an additional method or header or allow less headers, simply pass additional values to the parent constructor:

```
<?php
namespace OCA\MyApp\Controller;

use \OCP\AppFramework\ApiController;
use \OCP\IRequest;

class AuthorApiController extends ApiController {

   public function __construct($appName, IRequest $request) {
      parent::__construct(
         $appName,
         $request,
         'PUT, POST, GET, DELETE, PATCH',
         'Authorization, Content-Type, Accept',
         1728000);
   }

}
```

Hooks

Hooks are used to execute code before or after an event has occurred. This is for instance useful to run cleanup code after users, groups or files have been deleted. Hooks should be registered in the app.php:

```
<?php
namespace OCA\MyApp\AppInfo;

$app = new Application();
$app->getContainer()->query('UserHooks')->register();
```

The hook logic should be in a separate class that is being registered in the container

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Hooks\UserHooks;

class Application extends App {

   public function __construct(array $urlParams=array()){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('UserHooks', function($c) {
         return new UserHooks(
            $c->query('ServerContainer')->getUserManager()
         );
      });
   }
}
```

```php
<?php
namespace OCA\MyApp\Hooks;

class UserHooks {

   private $userManager;

   public function __construct($userManager){
      $this->userManager = $userManager;
   }

   public function register() {
      $callback = function($user) {
         // your code that executes before $user is deleted
      };
      $this->userManager->listen('\OC\User', 'preDelete', $callback);
   }

}
```

Available Hooks

The scope is the first parameter that is passed to the listen method, the second parameter is the method and the third one the callback that should be executed once the hook is being called, e.g.:

```php
<?php

// listen on user predelete
$callback = function($user) {
    // your code that executes before $user is deleted
};
$userManager->listen('\OC\User', 'preDelete', $callback);
```

Hooks can also be removed by using the removeListener method on the object:

```php
<?php

// delete previous callback
$userManager->removeListener(null, null, $callback);
```

The following hooks are available:

Session

Injectable from the ServerContainer by calling the method getUserSession().

Hooks available in scope \\OC\\User:

- preSetPassword (\OC\User\User $user, string $password, string $recoverPassword)
- postSetPassword (\OC\User\User $user, string $password, string $recoverPassword)
- preDelete (\OC\User\User $user)
- postDelete (\OC\User\User $user)
- preCreateUser (string $uid, string $password)
- postCreateUser (\OC\User\User $user)
- preLogin (string $user, string $password)
- postLogin (\OC\User\User $user)
- failedLogin (string $user)
- logout ()

UserManager

Injectable from the ServerContainer by calling the method getUserManager().

Hooks available in scope \\OC\\User:

- preSetPassword (\OC\User\User $user, string $password, string $recoverPassword)
- postSetPassword (\OC\User\User $user, string $password, string $recoverPassword)
- preDelete (\OC\User\User $user)
- postDelete (\OC\User\User $user)
- preCreateUser (string $uid, string $password)
- postCreateUser (\OC\User\User $user, string $password)

GroupManager

Hooks available in scope \\OC\\Group:

- preAddUser (\OC\Group\Group $group, \OC\User\User $user)
- postAddUser (\OC\Group\Group $group, \OC\User\User $user)
- preRemoveUser (\OC\Group\Group $group, \OC\User\User $user)
- postRemoveUser (\OC\Group\Group $group, \OC\User\User $user)
- preDelete (\OC\Group\Group $group)
- postDelete (\OC\Group\Group $group)
- preCreate (string $groupId)
- postCreate (\OC\Group\Group $group)

Filesystem Root

Injectable from the ServerContainer by calling the method getRootFolder(), getUserFolder() or getAppFolder().

Filesystem hooks available in scope \\OC\\Files:

- preWrite (\OCP\Files\Node $node)
- postWrite (\OCP\Files\Node $node)
- preCreate (\OCP\Files\Node $node)
- postCreate (\OCP\Files\Node $node)
- preDelete (\OCP\Files\Node $node)
- postDelete (\OCP\Files\Node $node)
- preTouch (\OCP\Files\Node $node, int $mtime)
- postTouch (\OCP\Files\Node $node)
- preCopy (\OCP\Files\Node $source, \OCP\Files\Node $target)
- postCopy (\OCP\Files\Node $source, \OCP\Files\Node $target)
- preRename (\OCP\Files\Node $source, \OCP\Files\Node $target)
- postRename (\OCP\Files\Node $source, \OCP\Files\Node $target)

Filesystem Scanner

Filesystem scanner hooks available in scope \\OC\\Files\\Utils\\Scanner:

- scanFile (string $absolutePath)
- scanFolder (string $absolutePath)
- postScanFile (string $absolutePath)
- postScanFolder (string $absolutePath)

Publishing in the ownCloud Marketplace

The ownCloud Marketplace

With the ownCloud marketplace, introduced in 2017, we offer a flexible and easy way to publish your apps and extend your ownCloud. In addition every ownCloud gets shipped with the new market app which makes it possible to manage apps directly out of your running ownCloud instance. Connected with the ownCloud marketplace it mirrors your marketplace account and provides an easy way to install and update

apps.

The process of publishing apps aims to be:

- Secure
- Transparent
- Welcoming
- Fair
- Easy to maintain

Apps in the store are divided into three levels of trust:

- Official
- Approved
- Experimental

With each level come requirements and a position in the store.

Official

Official apps are developed by and within the ownCloud community and its Github repository and offer functionality central to ownCloud. They are ready for serious use and can be considered a part of ownCloud.

Requirements:

- Developed in the ownCloud GitHub repo.
- Minimum of 2 active maintainers and contributions from others.
- Security audited and design reviewed.
- App is at least six months old and has seen regular releases.
- Follows app guidelines.
- Supports the same platforms and technologies mentioned in the release notes of the ownCloud version this app is made for.

ownCloud Marketplace:

- Available in Apps page in a separate category.
- Sorted first in all overviews, Official tag.
- Shown as featured on https://owncloud.org, etc.
- Major releases optionally featured on https://owncloud.org and sent to owncloud-announce list.
- New versions/updates approved by at least one other person.

Official apps include those that are part of the release tarball. We'd like to keep the tarball minimal, so most official apps are not part of the standard installation.

Approved

Approved apps are developed by trusted developers and have passed a cursory security check. They are actively maintained in an open code repository, and their maintainers deem them to be stable for casual to normal use.

Requirements:

- Code is developed in an open and version-managed code repository, ideally GitHub, with git. But other VCS' and hosting options are also OK.
- Minimum of one active developer/maintainer.
- Minimum 5 ratings, average score 60/100 or better.
- App is at least three months old.
- Follows app guidelines.
- The developer is trusted.
- App is subject to unannounced security audits.
- Has defined requirements and dependencies (like what browsers, databases, PHP versions and so on are supported).

Developer trust: The developer(s) is/are known in the community; he/she has/have been active for a while, have met others at events and/or worked with others in various areas.

Security audits: in practice, this means that at least some of the code of this developer has been audited; either through another app by the same developer or with an earlier version of the app. And that the attitude of the developer towards these audits has been positive.

ownCloud Marketplace:

- Visible in ownCloud Marketplace by default
- Sorted above experimental apps
- Search results sorted by ratings
- Developer can directly push new versions to the store
- Warning shows for security/stability risks

Experimental

Apps which have not been checked at all for security and/or are new, known to be unstable or under heavy development.

Requirements:

- No malicious intent found from this developer at any time
- 0 confirmed security problems
- Less than three unconfirmed `security flags'
- Rating over 20/100

ownCloud Marketplace:

- Show up in Apps page provided user has enabled allow installation of experimental apps in the settings.
- Warning about security and stability risks is shown for app
- Sorted below all others.

App Categories

The following categories are available for apps to be filed under:

- Automation
- Collaboration

- Customization
- External plugins
- Games
- Integration
- Multimedia
- Productivity
- Security
- Storage
- Tools

To make your app available under one of these categories, please make sure to user the proper tag in your info.xml:

```
<category>security</category>
```

Note: For publishing themes, this tag must be present but empty.

```
<category></category>
```

App Tags

Besides these categories apps can have different tags:

- Enterprise
- Verified
- Trusted

Enterprise

Apps with the Enterprise tag are official ownCloud enterprise apps. These can only be uploaded by ownCloud itself and represent ownCloud Enterprise Edition features.

ownCloud "Enterprise" tag

Verified

To get the verified label on your app, you must request a review. We then will look into your app and check if it meets the ownCloud app development guidelines (see below). The advantages of verified apps are that:

- they are labeled with verified badge.
- they are available in apps page in separate category.
- only verified apps can be displayed in the featured area.
- major releases optionally featured on https://owncloud.org and sent to the owncloud-announce list.

ownCloud "Verified" tag

Trusted

If your app reaches a rating level of 4 or higher based on 40 ratings or more it automatically gets the badge trusted. It represents a community oriented level of quality which makes it more attractive to other users. The advantages of trusted apps are that:

- they are labeled with trusted badge.
- the user can filter by trusted apps.

App Review Process

To request an app review go to Account > My Products > Edit app and click on the button Request review. Usually, it takes 3-5 work days to review your app. You will be notified about the result.

If it is successful, your app will get the verified badge. Please be aware of when uploading a new release to a verified app, you need to request a new review for the new release. To keep your verified badge, request the review before setting your new release to published.

App Guidelines

The following are the guidelines your app should follow to provide a high quality.

Legal and Security

- Apps can not use ownCloud in their name
- Irregular and unannounced security audits of all apps can and will take place.
- If any indication of malicious intent or bad faith is found the developer(s) in question can count on a minimum two-year ban from any ownCloud infrastructure.
- Malicious intent includes deliberate spying on users by leaking user data to a third party system or adding a back door (like a hard coded user account) to ownCloud. An unintentional security bug that gets fixed in time won't be considered bad faith.
- Apps do not violate any laws; it has to comply with copyright- and trademark law.
- App authors have to respond timely to security concerns and not make ownCloud more vulnerable to attack.

Distributing malicious or illegal applications can have legal consequences including, but not limited to ownCloud or affected users taking legal action.

Technical

- Apps can only use the public ownCloud API
- At time of the release of an app, it can only be configured to be compatible with the latest ownCloud release +1
- Apps should not cause ownCloud to break, consume excessive memory or slow ownCloud down
- Apps should not hamper functionality of ownCloud unless that is explicitly the goal of the app

Providing Information

When uploading an app, it should provide a professional and informative look and feel. To do so, please consider the following three points:

- The title of your app can be up to 50 characters. Provide a unique name, which

makes it easy for users to identify the product. Do not include your developer and/or company name in the title.

- The summary of your app can be up to 90 characters. Provide a short description. This will be displayed below the product titles.
- The description of your app can be up to 4000 characters and supports Markdown formatting. It should, ideally, provide all the necessary information about your app — especially information necessary to convince the user to download, use, and buy your app. So, don't get lost in technical details. Explain in simple, yet precise, steps what the user will get. When writing, focus on the benefits your app offers.

Images

- Provide meaningful images to your users.
- For best results, images should be 1400px wide and should go with a rough aspect ratio of 2:1
- The first image provided in your apps info.xml will be used as the preview image and is displayed in the top area of your marketplace app page.

Respect the Users

- Apps have to follow design and HTML/CSS layout guidelines
- Apps correctly clean up after themselves on uninstall and correctly handle up- and downgrades
- Apps communicate their intended purpose and active features, including features introduced through updates.
- Apps respect the users' choices and do not make unexpected changes, or limit users' ability to revert them. For example, they do not remove other apps or disable settings.
- Apps must respect user privacy. If user data is sent anywhere, this must be explained and be kept to a minimum for the functioning of an app. Use proper security measures when needed.

Disclaimer

ownCloud reserves the right to block and/or delete any uploaded app which does not comply with the ownCloud quality standards. Additionally, we reserve the right to ban publishers who attempt to upload malicious code. This does not depend on whether it happens intentionally or not.

Available Products Tags

Table 3. Available Product Tags

Tag	Description
id	A unique id. URL of your app will be based on this.
name	The name/title of your app; Max. 50 characters; Provide a concise name so users can identify your app easily; Do not include your developers/company name.
summary	Provide a short description (max. 90 chars). This gets displayed below the product title and on the product tiles; mandatory since ownCloud 10.0.0.

Tag	Description
description	Max. 4000 characters; Provide all necessary, detailed information about the product. This should contain all user relevant information. Don't get lost in technical details, focus on the benefits the product offers; Also, use markdown to layout your description.
license	At the moment following license are available: • OCL • ownCloud Commercial License This is for ownCloud Enterprise Apps **only** • AGPL • MIT License. To overwrite a release (using the same version number) it must be in state planned. Once published, you cannot replace a release.
category	The category you want to publish your app in; For all available categories see above.
screenshot	Image URL; insert multiple tags if you want to include multiple images; Note: marketplace will store images in its own file system. You do not need to provide the images on you own hosted area after the upload.

dependencies	Min and max version of ownCloud platform your app works with. For example:
	`<dependencies>`
	`<owncloud min-version="10.0" max-version="10.0" />`
	`</dependencies>`
	For a complete list of tags see: app/fundamentals/info.pdf.

Changelog

Breaking changes

8.2 RC2

The following breaking changes usually only affect applications which misuse existing API or do not follow best practices.

- The default Content-Security-Policy of AppFramework apps is now stricter but can be adjusted by developers. See https://github.com/owncloud/core/pull/13989
- Parameters passed to OC.generateUrl are now automatically encoded, this behavior can be adjusted by developers. See https://github.com/owncloud/core/pull/14266
- Views constructed by OCFilesView do not allow directory traversals anymore in the

constructor. See https://github.com/owncloud/core/pull/14342

- The CSRF token may now contain not URL compatible characters (for example the plus sign: +), developers have to ensure that the CSRF token is encoded properly before using it in URIs.
- The default RNG now returns all valid Base64 characters
- OC.msg escapes the message now by default (see https://github.com/owncloud/core/pull/14208)

Features

8.2 RC2

- There is a new OCSResponse and OCSController <controllers> which allows you to easily migrate OCS code to the App Framework. This was added purely for compatibility reasons and the preferred way of doing APIs is using a api
- You can now stream files in PHP by using the built in StreamResponse <controllers>.
- For more advanced use cases you can now implement the CallbackResponse <controllers> interface which allows your response to do its own response rendering
- Custom preview providers can now be implemented using OCP\IPreview::registerProvider
- There is a mightier class for remote web service requests at OCP\Http\Client
- OCP\\IImage allows now basic image manipulations such as resizing or rotating
- OCP\\Mail allows sending mails in an object-oriented way now
- OCP\\IRequest contains more methods now such as getting the request URI
- OCP\\Encryption allows writing custom encryption backends

Furthermore all public APIs have received a @since annotation allowing developers to see when a function has been introduced.

Deprecations

This is a deprecation roadmap which lists all current deprecation targets and will be updated from release to release. This lists the version when a specific method or class will be removed.

Deprecations on interfaces also affect the implementing classes!

Deprecation Policy

11.1

- OCP\\App::setActiveNavigationEntry has been deprecated in favour of \\OCP\\INavigationManager
- OCP\\BackgroundJob::registerJob has been deprecated in favour of OCP\\BackgroundJob\\IJobList
- OCP\\Contacts functions has been deprecated in favour of \\OCP\\Contacts\\IManager
- OCP\\DB functions have been deprecated in favour of the ones in \\OCP\\IDBConnection
- OCP\\Files::tmpFile has been deprecated in favour of \\OCP\\ITempManager::getTemporaryFile
- OCP\\Files::tmpFolder has been deprecated in favour of

\\OCP\\ITempManager::getTemporaryFolder

- \\OCP\\IServerContainer::getDb has been deprecated in favour of \\OCP\\IServerContainer::getDatabaseConnection
- \\OCP\\IServerContainer::getHTTPHelper has been deprecated in favour of \\OCP\\Http\\Client\\IClientService
- Legacy applications not using the AppFramework are now likely to use the deprecated OCP\\JSON and OCP\\Response code:
  - \\OCP\\JSON has been completely deprecated in favour of the AppFramework. Developers shall use the AppFramework instead of using the legacy OCP\\JSON code. This allows testable controllers and is highly encouraged.
  - \\OCP\\Response has been completely deprecated in favour of the AppFramework. Developers shall use the AppFramework instead of using the legacy OCP\\JSON code. This allows testable controllers and is highly encouraged.
- Diverse OCP\\Users function got deprecated in favour of OCP\\IUserManager:
  - OCP\\Users::getUsers has been deprecated in favour of OCP\\IUserManager::search
  - OCP\\Users::getDisplayName has been deprecated in favour of OCP\\IUserManager::getDisplayName
  - OCP\\Users::getDisplayNames has been deprecated in favour of OCP\\IUserManager::searchDisplayName
  - OCP\\Users::userExists has been deprecated in favour of OCP\\IUserManager::userExists
- Various static OCP\\Util functions have been deprecated:
  - OCP\\Util::linkToRoute has been deprecated in favour of \\OCP\\IURLGenerator::linkToRoute
  - OCP\\Util::linkTo has been deprecated in favour of \\OCP\\IURLGenerator::linkTo
  - OCP\\Util::imagePath has been deprecated in favour of \\OCP\\IURLGenerator::imagePath
  - OCP\\Util::isValidPath has been deprecated in favour of \\OCP\\IURLGenerator::imagePath

10.0

- An API added in one version of ownCloud only needs to be maintained as long as that version is not End of Life (EOL)
- An API can be removed completely in a future version of ownCloud if the release date of the version is later than the EOL date of the previous version
- Before removing an API completely, it needs to deprecated for at least a year. This is done by adding @deprecated tags.
- OCP\\IDb: This interface and the implementing classes will be removed in favor of OCP\\IDbConnection. Various layers in between have also been removed to be consistent with the PDO classes. This leads to the following changes:
  - Replace all calls on the db using getInsertId with lastInsertId
  - Replace all calls on the db using prepareQuery with prepare
  - The __construct method of OCP\\AppFramework\\Db\\Mapper no longer requires an instance of OCP\\IDb but an instance of OCP\\IDbConnection
  - The execute method on OCP\\AppFramework\\Db\\Mapper no longer returns an instance of OC_DB_StatementWrapper but an instance of PDOStatement

9.0

- The following methods have been moved into the OCP\\Template::<method> class instead of being namespaced directly:
  - OCP\\image_path
  - OCP\\mimetype_icon
  - OCP\\preview_icon
  - OCP\\publicPreview_icon
  - OCP\\human_file_size
  - OCP\\relative_modified_date
  - OCP\\html_select_options
- OCP\\simple_file_size has been deprecated in favour of OCP\\Template::human_file_size
- The OCP\\PERMISSION_<permission> and OCP\\FILENAME_INVALID_CHARS have been moved to OCP\\Constants::<old name>
- The OC_GROUP_BACKEND_<method> and OC_USER_BACKEND_<method> have been moved to OC_Group_Backend::<method> and OC_User_Backend::<method> respectively

8.3

- OCP\AppFramework\IApi: full class
- OCP\AppFramework\IAppContainer: methods getCoreApi and log
- OCP\AppFramework\Controller: methods params, getParams, method, getUploadedFile, env, cookie, render

8.1

- \OC\Preferences and \OC_Preferences

Market App

Since ownCloud X (10.0.0) every ownCloud instance gets shipped with the market app. This app makes it easy to manage your applications out of the box. To connect your market app with the ownCloud Marketplace:

- Get you API key under My Account
- Inside the market app go to Settings
- Paste your API key and click on Save

You are now able to maintain any app in downloads/installations/updates from your ownCloud installation directly.

ownCloud Instances in Protected Environments (DMZ)

To use the market app your ownCloud instance must have an internet connection. If your instance is running in a protected environment (DMZ or similar) you cannot use the market app. You need to upload the apps manually in this case. Every app can be downloaded manually from the marketplace.

Application Development - Advanced Details

In this section, you will find the advanced details for developing an ownCloud application.

Custom Filesystem Caches

The metadata cache in ownCloud can be overridden by a storage class backend which implements the following methods:

Method	Description
getCache($path = `, $storage = null) getScanner($path = `, $storage = null) getWatcher($path = `, $storage = null)	For overwriting the cache itself. For overwriting the meta data scanning behavior. For overwriting the behavior of checking for external changes.

It's unlikely that an app will need to override any of the three systems; as long as a storage backend behaves accordingly, the cache systems will work on any storage backend.

But, here are some cases where it may be practical to do so:

- **Overriding the cache:** This may be helpful in the case of shared storage. In this case, the overriding class should redirect any cache operation to the cache of the user that owns the share.
- **Overriding the scanner:** This is useful in cases where it would provide an efficient way to retrieve the metadata of a significant number of files and folders. In doing so it avoids the need to perform a large number of round-trip requests.
- **Overriding the watcher:** This could be useful for changing the behavior for detecting changes made to a storage from outside ownCloud.

However — in almost all cases — overriding the hasUpdated() method of a storage provides sufficient flexibility.

If any of these three systems need to be overridden, one of the following classes should be sub-classed:

- \OC\Files\Cache\Cache
- \OC\Files\Cache\Scanner
- \OC\Files\Cache\Watcher

This class should then return the subclass from one of the three methods listed above.

Cache

Instead of creating a full, custom, cache object, you can also use the same wrapper pattern as when creating custom storage backends. Cache wrappers should be implemented by overriding the getCache() method. In addition, it may also be useful to override the following methods:

Method	Description
get($file)	Returns either the cache entries for a file or folder or false if the file is not in the cache.
getFolderContents($path)	Returns the cache entries for all files and folders in a folder or an empty array if the folder is not in the cache.
getFolderContentsById($id)	Same as getFolderContents(), but it uses a file id instead of a path.

Method	Description
put($file, $data)	Saves a cache entry for a file. If the file is already in the cache then update() is called automatically.
update($id, $data)	Updates an existing cache entry. Only the changed values need to be provided in $data, any omitted values will remain unchanged.
getId($path)	Retrieves the file id for a file or folder. A file id is a numeric id for a file or folder that's unique within an ownCloud instance which stays the same for the lifetime of a file even through renaming.
getParentId($path)	Retrieves the file id of the parent folder or =1 if the file has no parent, root, entry.
inCache($file)	Checks if a file is in the cache.
remove($file)	Removes a file or folder from the cache. In the case of removing a folder, it should remove all child entries as well.
move($source, $target)	Renames a file or folder in the cache. In the case of moving a folder, it should also move all child entries.
moveFromCache($sourceCache, $sourcePath, $targetPath)	Moves a file or folder from a cache instance to a local path.
clear()	Removes all entries from the cache.
getStatus($file)	Retrieves the scanned status of a file or folder.
search($pattern)	Searches the cache for a file or folder where the filename matches $pattern. SQL style wildcards are used in the pattern.
searchByMime($mimetype)	Searches for a file or folder with a matching mimetype. Both full mimetypes (`text/plain') and mimetype groups (`text') should be supported as search option.
correctFolderSize($path)	Recalculates the size of a folder and all parent folders.
calculateFolderSize($path)	Recalculates the size of a single folder.
getAll()	Retrieves the file id for all files and folder in the cache
getIncomplete()	Retrieve folders which have a status of Cache::SHALLOW.
getPathById($id)	Retrieve the path of a file or folder whose file id matches $id. Returns null if a match is not found.
static getById($id)	Retrieves the path and storage id for a file whose file id matches $id. This is deprecated in favor of getPathById().

Cache Entries

A cache entry is an associative array that should contain, at least, the following values:

Method	Type	Description
fileid	int	The numeric id of a file (see getId(), above).
storage	int	The numeric id of the storage the file is stored on.
path	string	The path of the file within the storage (e.g., `foo/bar.txt').
name	string	The basename of a file or folder (’bar.txt).
mimetype	string	The full mimetype of the file (e.g., `text/plain').
mimepart	string	The mimetype group (e.g., `text').
size	int	The size of the file or folder in bytes.
mtime	int	The last modified date of the file as a UNIX timestamp as shown in the UI.
storage_mtime	int	The last modified date of the file as a UNIX timestamp as stored on the storage.

Note that when a file is updated ownCloud also updates the modification time of **all** parent folders. Doing so makes it visible to the user exactly which folder has most recently been updated. However, ownCloud’s modification time can differ from the mtime value on the underlying storage. But, this usually only changes when a direct child is added, removed, or renamed.

Method	Type	Description
etag	string	An Etag is used to detect changes to files and folders. An Etag of a *file* changes whenever the content of the file changes. An Etag of a *folder* changes whenever a file *in* the folder has changed.
permissions	int	The permissions for the file. These are stored as a bitwise combination of \OCP\PERMISSION_READ, \OCP\PERMISSION_CREATE, \OCP\PERMISSION_UPDATE, \OCP\PERMISSION_DELETE, and \OCP\PERMISSION_SHARE.

CacheWrappers

Just like storage wrappers, cache wrappers can be used to change the behavior of an existing cache. ownCloud comes with two cache wrappers which can be useful for applications; these are:

- \OC\Files\Cache\Wrapper\CacheJail
- \OC\Files\Cache\Wrapper\CachePermissionsMask

These serve the same purpose as the two similarly named storage wrappers. Implementing a cache wrapper can be done by sub-classing \OC\Files\Cache\CacheWrapper. Inside this class, the wrapped cache will be available as $this→cache.

Besides providing the options to override any method of the wrapped cache, the cache wrapper also provides the convenience method formatCacheEntry($entry). This can be overridden to allow for easier changes to any method that returns cache entries.

Scanner

It might be useful to override the following methods of the scanner:

Method	Description
getData($path)	Retrieves all metadata of a path to put in the cache. It returns an array which should contain the following keys: mimetype, mtime, size, etag, storage_mtime\, and`permissions`.size`should always being-1` for folders.
scanFile($file)	Scans a single file, or scans a folder by passing self::SCAN_RECURSIVE (or true) as the second parameter. When scanning folders, the scanner should recurse into any sub-directory and the size of any folder should be calculated correctly. If not, the scanner should only scan the direct children of the folder. Any folder that's not fully scanned should have it's size set to -1.
backgroundScan()	Should do a recursive scan on all folders which have not previously been fully scanned. The size should be set to -1.

Watcher

The watcher is responsible for checking for outside changes made to the filesystem and updating the cache accordingly. As noted above, in most cases overriding the hasUpdated() method of a storage backend sub-class is sufficient. However, the following methods could be overridden, if necessary:

Method	Description
checkUpdate($path)	Checks if a file or folder has been changed externally. If so it updates the cache and return true, else return false.
cleanFolder($path)	Checks a folder for any child entries that are no longer in the storage. This should be called automatically by checkUpdate() if that method detects an update.

An app or admin can also change the watcher behavior by setting it's policy by calling setPolicy($policy). This method can take the following values:

Method	Description
Watcher::CHECK_NEVER	Don't check for any external change. This is recommended if you're certain that no outside changes will be made.
Watcher::CHECK_ONCE	Check each path for updates at most once during a request (default).
Watcher::CHECK_ALWAYS	Check for external changes any number of times during a request. It is mostly useful for unit tests.

Updater

Another cache related system, which developers should be aware of when working with custom caches, is the updater. The updater (\OC\Files\Cache\Updater) is responsible for updating the cache when any change is made from inside ownCloud. It will call either the scanner or the cache of a storage to make the required changes. The updater **can not** be overwritten by storage backends.

Create Custom Storage Backends

The preferred way for applications to create new storage backends is to create a subclass of \OC\Files\Storage\Common and implement the abstract methods. It's also possible to create storage backends by implementing the required interface.

However, by sub-classing the common backend a lot of the boiler plate is taken care of. What's more, it provides common implementations and fallbacks to reduce the amount of work it is to create a storage backend.

Required Methods

All storage backends sub-classing the common storage backend must implement the following methods:

Method	Description
mkdir($path)	Creates a new folder on the storage.
rmdir($path)	Deletes an existing folder on the storage.
opendir($path)	Opens a directory handle.
stat($path)	Retrieves the metadata for the file or folder. The returned array should, at least, contain mtime and size.
filetype($path)	Returns the file type; either file or dir.
file_exists($path)	Checks if a file or folder exists.
unlink($path)	Removes a file or folder. This isn't only for deleting files, unlike PHP's unlink method.
fopen($path, $mode)	Opens a file handle for a file
touch($path, $mtime = null)	Updates the mtime of a file or folder. If $mtime is omitted the current time should be used.

Suggested Methods

The common storage backends provide fallback implementations for a number of methods to make them easier to implement. However, some of fallback implementations are either inefficient or don't always provide the correct result for custom storage backends. Given that, please consider overriding one or more of the following methods:

Method	Description
rename($sourcePath, $targetPath)	Renames a file. The default implementation uses copy and unlink which is very inefficient.
copy($sourcePath, $targetPath)	Copies a file. The default implementation copies using streams. This is inefficient for remote storages as it downloads and re-uploads the file.
isReadable($path)	Checks if a file is readable. It defaults to true if the file exists.
isUpdatable($path)	Checks if a file or folder can be updated. This includes being written to or renamed. It defaults to true if the file exists.

Method	Description
isCreatable($path)	Checks if new files can be created in a folder It defaults to isUpdatable($path).
isDeletable($path)	Checks if a file can be deleted. It defaults to isUpdatable($path).
isSharable($path)	Checks if a file can be shared. It defaults to isReadable($path).
free_space($path)	Checks the free space on the storage in bits.

Other Useful Methods

The default implementation for the following methods are good for most storage backends. But, providing an alternate implementation *can* improve user experience.

Method	Description
file_put_contents($path, $data)	Stores a file on the storage. It defaults to using fopen($path, 'w').
file_get_contents($path)	Retrieves a file from storage. Defaults to using fopen($path, 'r').
getMimeType($path)	Retrieves the mimetype of a file or folder. Defaults to guessing the mimetype from the extension. The mimetype of a folder is _[required] to be 'httpd/unix-directory'.
hasUpdated($path, $time)	Checks if a file or folder has been updated since $time. If you're certain the files on the storage will not be updated outside of ownCloud you can always return false to increase performance.
getETag($path)	Retrieves the Etag for a file or folder.
verifyPath($path, $fileName)	Checks if a filename is valid for the storage backend. It defaults to checking for invalid characters or names for the server platform.

Copying and Moving Between Storage Backends

When copying or moving files between different storages a stream copy is used by default. This works well for copying between different types of storages, such as from local to SMB. But, there are cases where a more efficient copy is possible, such as between two SMB storages on the same server. In these cases, storage backends can override the cross-storage behavior by overriding the following methods:

- copyFromStorage(\OCP\Files\Storage $sourceStorage, $sourceInternalPath, $targetInternalPath, $preserveMtime = false);
- moveFromStorage(\OCP\Files\Storage $sourceStorage, $sourceInternalPath ,$targetInternalPath);

Working With Streams

Both fopen() and opendir() require storage backends to return native PHP streams for maximum compatibility. ownCloud comes with several classes which make it easier for storage backends to create native PHP streams for backends not supported by PHP's own streamWrapper.

IteratorDirectory

Icewind\Streams\IteratorDirectory allows for creating a directory handle from an array or iterator.

```
$fileNames = $this->getFolderContentsSomehow();
return IteratorDirectory::wrap($fileNames);
```

CallbackWrapper

Icewind\Streams\CallbackWrapper wraps an existing file handle, and allows for hooking into file reads and writes, and closing streams. The most common use case for this class in storage backends is for implementing fopen() with writable streams. This is because writing to and closing streams happens outside the storage implementation. As a result, the storage backend needs a way to upload the changed file back to the backend. This can be done by attaching a close-callback to a stream for a temporary file.

```
$tempFile = $this->downloadFile($path);
$handle = fopen($tempFile, $mode);
return CallBackWrapper::wrap($handle, null, null function() use ($path,
$tempFile) {
    $this->uploadFile($tempFile, $path);
    unlink($tempFile);
}
```

Storage Wrappers

Besides implementing a complete custom storage backend, ownCloud allows for modifying the behavior of an existing storage by applying a wrapper to it. Storage wrappers need to implement the full storage API methods. Examples of storage wrappers are

- **The Quota wrapper.** This changes the behavior of free_space by limiting the free space returned by the wrapped storage to a configured maximum
- **The Encryption wrapper**. This encrypts and decrypts the data on the fly by overwriting file_put_contents, file_get_contents, and fopen.

When implementing a storage wrapper, the wrapped storage is available as $this→storage. Storage wrappers can either be applied globally to all used storages using \OC\Files\Filesystem::addStorageWrapper($name, $wrapper) or to a specific storage, while mounting the storage from the app. Implementing a storage wrapper is done by sub-classing \OC\Files\Storage\Wrapper\Wrapper and overwriting any of its methods.

Global Storage Wrappers

For using a storage wrapper globally, you provide a callback which will be called for each used storage. The callback can than determine if a wrapper should be applied to the given storage, based on the storage or mountpoint, or whether it needs to return the storage unwrapped.

```
Filesystem::addStorageWrapper('fooWrapper', function($mountPoint, $storage) {
    if ($storage->instanceOfStorage('FooStorage')) {
        return new FooWrapper(['storage' => $storage]);
    } else {
        return $storage;
    }
}
```

Wrappers for a Single Storage

Sometimes an app can avoid having to create a custom storage backend by instead modifying the behavior of an existing one. ownCloud comes with a few generic storage wrappers which might be useful when doing so, which include PermissionsMask and Jail.

PermissionsMask

\OC\Files\Storage\Wrapper\PermissionsMask can be used to restrict the permissions on an existing storage. A sample use case is to create a read-only ftp backend.

```
$storage = $this->createStorageToWrapSomehow();
return new PermissionsMask([
  'storage' => $storage,
  'mask' => \OCP\Constant::PERMISSION_READ | \OCP\Constant
::PERMISSION_SHARE
]);
```

Jail

\OC\Files\Storage\Wrapper\Jail can be used to limit storage interaction to a sub-folder of an existing storage.

```
$storage = $this->createStorageToWrapSomehow();
return new Jail([
  'storage' => $storage,
  'root' => 'some/folder/in/the/storage'
]);
```

A Note on instanceof()

Since storage wrappers wrap an existing storage instead of sub-classing it, it is not possible to determine if the storage is a specific class using PHP's instanceof operator. Instead, you need to call the instanceOfStorage() method on the class with the fully-qualified class name.

```
// Only works if no wrappers are applied
if ($storage instanceof \OC\Files\Storage\DAV) {
    // ...
}

// Works regardless of any wrapper
if ($storage->instanceOfStorage('\OC\Files\Storage\DAV')) {
    // ...
}
```

instanceOfStorage() can also be used to check if a certain wrapper is applied to a storage.

Mounting Storages

For an app to add its storages to the filesystem it should implement a mount provider and register it with the filesystem. Implementing mount providers is done by implementing the \OCP\Files\Config\IMountProvider interface, containing the getMountsForUser(IUser $user, IStorageFactory $storageFactory) method, which returns a list of mountpoints that should be created for a user.

```
class MyMountProvider implements IMountProvider {
    public function getMountsForUser(IUser $user, IStorageFactory $loader) {
        $config = magicallyGetMountConfigurations();
        return array_map(function($mountOptions) use ($loader) {
            return new Mount(
                $mountOptions['class'],
                $mountOptions['mountPoint'],
                $mountOptions['storageOptions'],
                $loader
            );
        }, $config);
    }
}
```

Registering a mount provider should be done from an app's appinfo/app.php. Note that any mount provider registered after the filesystem is setup for a user will not be called again for that user.

```
$provider = new MyMountProvider();
\OC::$server->getMountProviderCollection()
        ->registerProvider($provider);
```

Storage Backends

External Storage Backends

This section shows how a standard app can provide external storage backends. To do so, requires several steps. These are:

- Configure the filesystem type

- Implement the storage class(es)
- Create the backend adapter
- Register the backend adapter
- Test the storage backend

To save time, however, you can learn from an existing example, by reading through the source code of the FTP external storage app.

Configure the Filesystem Type

First, the /appinfo/info.xml must be adjusted to specify the type as: filesystem. For example:

Implement the Storage Class(es)

Next, you need to create a storage class. Usually, you should implement the interface \\OCP\\Files\\Storage\\IStorage. But, the easiest way is to directly extend \\OCP\\Files\\Storage\\StorageAdapter, as it already provides an implementation for many of the commonly required methods.

Here's an example of how you would create one that implements all the filesystem operations required by ownCloud, using a fictitious library called FakeStorageLib.

For this example we mapped the available storage methods to the ones from the library. Note that, in many cases, the underlying library might not support some operations and might need extra code to work this around.

When extending StorageAdapter, it is good practice to implement the following methods, for performance reasons:

- file_exists
- filetype
- fopen
- getId
- mkdir
- opendir
- rmdir
- stat
- touch
- unlink

If you don't, your storage backend will still work. But, it will likely not perform as well as it could. In the case of the rename method, this is because it uses a combination of a stream copy plus a delete for renaming a file.

Stat/Metadata Cache

To create a mature implementation, we need to consider stat and metadata caching. Within a single PHP request, ownCloud might call the same storage methods repeatedly, due to different checks which it needs to carry out. As a result, there is the potential to incur significant overhead, when working with the underlying filesystem.

To avoid — or at the very least *reduce* this — a stat/metadata cache should be implemented, if the underlying library does not support stat/metadata caching. To do this, the metadata of any folder entries which are read should be cached in a local

array and returned by the storage class' methods.

Writing a Flysystem Adapter

Instead of writing everything by hand, it is also possible to write an ownCloud adapter based on a Flysystem adapter, as external storage. You can see how it was done in the FTP storage adapter.

Create the Backend Adapter

After implementing the storage class, a backend adapter needs to be created. To do that, create a class that extends from \\OCP\\Files\\External\\Backend:

Definition Parameters

Flags

Flag	Description
DefinitionParameter::FLAG_NONE	No flags (default)
DefinitionParameter::FLAG_OPTIONAL	For optional parameters

Types

Type	Description
DefinitionParameter::VALUE_TEXT	Text field (default)
DefinitionParameter::VALUE_PASSWORD	Masked text field, for passwords and
	keys
DefinitionParameter::VALUE_BOOLEAN	Boolean / checkbox
DefinitionParameter::VALUE_HIDDEN	Hidden field, useful with custom
	scripts

Authentication Schemes

Several authentication schemes can be specified.

Scheme	Description
AuthMechanism::SCHEME_NULL	No authentication supported
AuthMechanism::SCHEME_BUILTIN	Authentication is provided through
	definition parameters
AuthMechanism::SCHEME_PASSWORD	Support for password-based auth,
	provides two fields user and
	password to the parameter list
AuthMechanism::SCHEME_OAUTH1	OAuth1, provides fields app_key,
	app_secret, token, token_secret

Scheme	Description
	and configured
AuthMechanism::SCHEME_OAUTH2	OAuth2, provides fields client_id,
	client_secret, token and configured
AuthMechanism::SCHEME_PUBLICKEY	Public key, provides fields user,
	public_key, private_key

Custom User Interface

When dealing with complex field values or workflows like OAuth, an application might need to provide custom JavaScript code to implement such workflow. To add a custom script, use the following in the backend constructor:

```
$this->addCustomJs('script');
```

This will automatically load the script /js/script.js from the app folder. The script itself will need to inject events into the external storage GUI as there is currently no proper public API to do so.

Register the Backend Adapter

With the backend adapter created, it next needs to be registered. This can be done in the Application class by implementing the IBackendProvider interface, as in the example below:

```
:include: examples/storage-backend/OCA/MyStorageApp/AppInfo/Application.php
```

Then in appinfo/app.php instantiate the Application class:

```
<?php

$app = new \OCA\MyStorageApp\AppInfo\Application();
```

Test the Storage Backend

Once the steps above are done, you should be able to mount the storage in the external storage section.

Create Custom Storage Backends

The preferred way for applications to create new storage backends is to create a subclass of \OC\Files\Storage\Common and implement the abstract methods. It's also possible to create storage backends by implementing the required interface.

However, by sub-classing the common backend a lot of the boiler plate is taken care of. What's more, it provides common implementations and fallbacks to reduce the amount of work it is to create a storage backend.

Required Methods

All storage backends sub-classing the common storage backend must implement the following methods:

Method	Description
mkdir($path)	Creates a new folder on the storage.
rmdir($path)	Deletes an existing folder on the storage.
opendir($path)	Opens a directory handle.
stat($path)	Retrieves the metadata for the file or folder. The returned array should, at least, contain mtime and size.
filetype($path)	Returns the file type; either file or dir.
file_exists($path)	Checks if a file or folder exists.
unlink($path)	Removes a file or folder. This isn't only for deleting files, unlike PHP's unlink method.
fopen($path, $mode)	Opens a file handle for a file
touch($path, $mtime = null)	Updates the mtime of a file or folder. If $mtime is omitted the current time should be used.

Suggested Methods

The common storage backends provide fallback implementations for a number of methods to make them easier to implement. However, some of fallback implementations are either inefficient or don't always provide the correct result for custom storage backends. Given that, please consider overriding one or more of the following methods:

Method	Description
rename($sourcePath, $targetPath)	Renames a file. The default implementation uses copy and unlink which is very inefficient.
copy($sourcePath, $targetPath)	Copies a file. The default implementation copies using streams. This is inefficient for remote storages as it downloads and re-uploads the file.
isReadable($path)	Checks if a file is readable. It defaults to true if the file exists.
isUpdatable($path)	Checks if a file or folder can be updated. This includes being written to or renamed. It defaults to true if the file exists.
isCreatable($path)	Checks if new files can be created in a folder It defaults to isUpdatable($path).
isDeletable($path)	Checks if a file can be deleted. It defaults to isUpdatable($path).
isSharable($path)	Checks if a file can be shared. It defaults to isReadable($path).
free_space($path)	Checks the free space on the storage in bits.

Other Useful Methods

The default implementation for the following methods are good for most storage backends. But, providing an alternate implementation *can* improve user experience.

Method	Description
file_put_contents($path, $data)	Stores a file on the storage. It defaults to using fopen($path, 'w').
file_get_contents($path)	Retrieves a file from storage. Defaults to using fopen($path, 'r').
getMimeType($path)	Retrieves the mimetype of a file or folder. Defaults to guessing the mimetype from the extension. The mimetype of a folder is _[required] to be 'httpd/unix-directory'.
hasUpdated($path, $time)	Checks if a file or folder has been updated since $time. If you're certain the files on the storage will not be updated outside of ownCloud you can always return false to increase performance.
getETag($path)	Retrieves the Etag for a file or folder.
verifyPath($path, $fileName)	Checks if a filename is valid for the storage backend. It defaults to checking for invalid characters or names for the server platform.

Copying and Moving Between Storage Backends

When copying or moving files between different storages a stream copy is used by default. This works well for copying between different types of storages, such as from local to SMB. But, there are cases where a more efficient copy is possible, such as between two SMB storages on the same server. In these cases, storage backends can override the cross-storage behavior by overriding the following methods:

- copyFromStorage(\OCP\Files\Storage $sourceStorage, $sourceInternalPath, $targetInternalPath, $preserveMtime = false);
- moveFromStorage(\OCP\Files\Storage $sourceStorage, $sourceInternalPath ,$targetInternalPath);

Working With Streams

Both fopen() and opendir() require storage backends to return native PHP streams for maximum compatibility. ownCloud comes with several classes which make it easier for storage backends to create native PHP streams for backends not supported by PHP's own streamWrapper.

IteratorDirectory

Icewind\Streams\IteratorDirectory allows for creating a directory handle from an array or iterator.

```
$fileNames = $this->getFolderContentsSomehow();
return IteratorDirectory::wrap($fileNames);
```

CallbackWrapper

Icewind\Streams\CallbackWrapper wraps an existing file handle, and allows for hooking into file reads and writes, and closing streams. The most common use case for this class in storage backends is for implementing fopen() with writable streams. This is because writing to and closing streams happens outside the storage implementation. As a result, the storage backend needs a way to upload the changed file back to the backend. This can be done by attaching a close-callback to a stream for a temporary file.

```
$tempFile = $this->downloadFile($path);
$handle = fopen($tempFile, $mode);
return CallBackWrapper::wrap($handle, null, null function() use ($path,
$tempFile) {
    $this->uploadFile($tempFile, $path);
    unlink($tempFile);
}
```

Storage Wrappers

Besides implementing a complete custom storage backend, ownCloud allows for modifying the behavior of an existing storage by applying a wrapper to it. Storage wrappers need to implement the full storage API methods. Examples of storage wrappers are

- **The Quota wrapper.** This changes the behavior of free_space by limiting the free space returned by the wrapped storage to a configured maximum
- **The Encryption wrapper**. This encrypts and decrypts the data on the fly by overwriting file_put_contents, file_get_contents, and fopen.

When implementing a storage wrapper, the wrapped storage is available as $this→storage. Storage wrappers can either be applied globally to all used storages using \OC\Files\Filesystem::addStorageWrapper($name, $wrapper) or to a specific storage, while mounting the storage from the app. Implementing a storage wrapper is done by sub-classing \OC\Files\Storage\Wrapper\Wrapper and overwriting any of its methods.

Global Storage Wrappers

For using a storage wrapper globally, you provide a callback which will be called for each used storage. The callback can than determine if a wrapper should be applied to the given storage, based on the storage or mountpoint, or whether it needs to return the storage unwrapped.

```
Filesystem::addStorageWrapper('fooWrapper', function($mountPoint, $storage) {
    if ($storage->instanceOfStorage('FooStorage')) {
        return new FooWrapper(['storage' => $storage]);
    } else {
        return $storage;
    }
}
```

Wrappers for a Single Storage

Sometimes an app can avoid having to create a custom storage backend by instead modifying the behavior of an existing one. ownCloud comes with a few generic storage wrappers which might be useful when doing so, which include PermissionsMask and Jail.

PermissionsMask

\OC\Files\Storage\Wrapper\PermissionsMask can be used to restrict the permissions on an existing storage. A sample use case is to create a read-only ftp backend.

```
$storage = $this->createStorageToWrapSomehow();
return new PermissionsMask([
  'storage' => $storage,
  'mask' => \OCP\Constant::PERMISSION_READ | \OCP\Constant
::PERMISSION_SHARE
]);
```

Jail

\OC\Files\Storage\Wrapper\Jail can be used to limit storage interaction to a sub-folder of an existing storage.

```
$storage = $this->createStorageToWrapSomehow();
return new Jail([
  'storage' => $storage,
  'root' => 'some/folder/in/the/storage'
]);
```

A Note on instanceof()

Since storage wrappers wrap an existing storage instead of sub-classing it, it is not possible to determine if the storage is a specific class using PHP's instanceof operator. Instead, you need to call the instanceOfStorage() method on the class with the fully-qualified class name.

```
// Only works if no wrappers are applied
if ($storage instanceof \OC\Files\Storage\DAV) {
   // ...
}

// Works regardless of any wrapper
if ($storage->instanceOfStorage('\OC\Files\Storage\DAV')) {
   // ...
}
```

instanceOfStorage() can also be used to check if a certain wrapper is applied to a storage.

Mounting Storages

For an app to add its storages to the filesystem it should implement a mount provider and register it with the filesystem. Implementing mount providers is done by implementing the \OCP\Files\Config\IMountProvider interface, containing the getMountsForUser(IUser $user, IStorageFactory $storageFactory) method, which returns a list of mountpoints that should be created for a user.

```
class MyMountProvider implements IMountProvider {
   public function getMountsForUser(IUser $user, IStorageFactory $loader) {
      $config = magicallyGetMountConfigurations();
      return array_map(function($mountOptions) use ($loader) {
         return new Mount(
            $mountOptions['class'],
            $mountOptions['mountPoint'],
            $mountOptions['storageOptions'],
            $loader
         );
      }, $config);
   }
}
```

Registering a mount provider should be done from an app's appinfo/app.php. Note that any mount provider registered after the filesystem is setup for a user will not be called again for that user.

```
$provider = new MyMountProvider();
\OC::$server->getMountProviderCollection()
         ->registerProvider($provider);
```

External Storage Backends

This section shows how a standard app can provide external storage backends.

To do so, requires several steps. These are:

- Configure the filesystem type
- Implement the storage class(es)
- Create the backend adapter
- Register the backend adapter
- Test the storage backend

To save time, however, you can learn from an existing example, by reading through the source code of the FTP external storage app.

Configure the filesystem type

First, the /appinfo/info.xml must be adjusted to specify the type as filesystem.
For example:

```
<?xml version="1.0"?>
<info>
 <id>mystorageapp</id>
 <name>My storage app</name>
 ...
 <types>
  <filesystem/>
 </types>
 ...
</info>
```

Implement the storage class(es)

Next, you need to create a storage class. Usually, you should implement the interface \\OCP\\Files\\Storage\\IStorage. But, the easiest way is to directly extend \\OCP\\Files\\Storage\\StorageAdapter, as it already provides an implementation for many of the commonly required methods.

Here's an example of how you would create one that implements all the filesystem operations required by ownCloud, using a fictitious library called FakeStorageLib.

For this example we mapped the available storage methods to the ones from the library. Note that, in many cases, the underlying library might not support some operations and might need extra code to work this around.

When extending StorageAdapter, it is good practice to implement the following methods, for performance reasons:

- file_exists
- filetype
- fopen
- getId
- mkdir
- opendir
- rmdir
- stat
- touch
- unlink

If you don't, your storage backend will still work. But, it will likely not perform as well as it could. In the case of the rename method, this is because it uses a combination of a stream copy plus a delete for renaming a file.

Stat/metadata cache

To create a mature implementation, we need to consider stat and metadata caching. Within a single PHP request, ownCloud might call the same storage methods repeatedly, due to different checks which it needs to carry out. As a result, there is the potential to incur significant overhead, when working with the underlying filesystem.

To avoid — or at the very least *reduce* this — a stat/metadata cache should be implemented, if the underlying library does not support stat/metadata caching. To do this, the metadata of any folder entries which are read should be cached in a local

array and returned by the storage class' methods.

Writing a Flysystem adapter

Instead of writing everything by hand, it is also possible to write an ownCloud adapter based on a Flysystem adapter, as external storage. You can see how it was done in the FTP storage adapter.

Create the backend adapter

After implementing the storage class, a backend adapter needs to be created. To do that, create a class that extends from \\OCP\\Files\\External\\Backend:

Definition parameters

Flags:

Flag	Description
DefinitionParameter::FLAG_NONE	No flags (default)
DefinitionParameter::FLAG_OPTIONAL	For optional parameters

Types:

Type	Description
DefinitionParameter::VALUE_TEXT	Text field (default)
DefinitionParameter::VALUE_PASSWORD	Masked text field, for passwords and
	keys
DefinitionParameter::VALUE_BOOLEAN	Boolean / checkbox
DefinitionParameter::VALUE_HIDDEN	Hidden field, useful with custom
	scripts

Authentication schemes

Several authentication schemes can be specified.

Scheme	Description
AuthMechanism::SCHEME_NULL	No authentication supported
AuthMechanism::SCHEME_BUILTIN	Authentication is provided through
	definition parameters
AuthMechanism::SCHEME_PASSWORD	Support for password-based auth,
	provides two fields user and
	password to the parameter list
AuthMechanism::SCHEME_OAUTH1	OAuth1, provides fields app_key,
	app_secret, token, token_secret

Scheme	Description
	and configured
AuthMechanism::SCHEME_OAUTH2	OAuth2, provides fields client_id,
	client_secret, token and configured
AuthMechanism::SCHEME_PUBLICKEY	Public key, provides fields user,
	public_key, private_key

Custom user interface

When dealing with complex field values or workflows like OAuth, an application might need to provide custom JavaScript code to implement such workflow. To add a custom script, use the following in the backend constructor:

```
$this->addCustomJs('script');
```

This will automatically load the script /js/script.js from the app folder. The script itself will need to inject events into the external storage GUI as there is currently no proper public API to do so.

Register the backend adapter

With the backend adapter created, it next needs to be registered. This can be done in the Application class by implementing the IBackendProvider interface, as in the example below:

```php
<?php

namespace OCA\MyStorageApp\AppInfo;

use OCP\AppFramework\App;
use OCP\AppFramework\IAppContainer;
use OCP\IContainer;
use OCP\Files\External\Config\IBackendProvider;

/**
 * @package OCA\MyStorageApp\AppInfo
 */
class Application extends App implements IBackendProvider {
	public function __construct(array $urlParams = array()) {
		parent::__construct('mystorageapp', $urlParams);
		$container = $this->getContainer();

		// retrieve the backend service
		$backendService = $container->getServer()->getStoragesBackendService();

		// register this class as backend provider
		$backendService->registerBackendProvider($this);
	}

	/**
	 * Return a list of backends to register
	 */
	public function getBackends() {
		$container = $this->getContainer();
		$backends = [
			$container->query('OCA\MyStorageApp\Backend\MyStorageBackend'),
		];
		return $backends;
	}
}
```

Then in appinfo/app.php instantiate the Application class:

```php
<?php

 $app = new \OCA\MyStorageApp\AppInfo\Application();
```

Test the storage backend

Once the steps above are done, you should be able to mount the storage in the external storage section.

Translation

ownCloud's translation system is powered by Transifex. To start translating sign up and enter a group. If translations for your community app should be added to Transifex follow the steps decribed at the end of this page.

PHP

Should it ever be needed to use localized strings on the server-side, simply inject the L10N service from the ServerContainer into the needed constructor

```
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Service\AuthorService;

class Application extends App {

   public function __construct(array $urlParams=array()){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('AuthorService', function($c) {
         return new AuthorService(
            $c->query('L10N')
         );
      });

      $container->registerService('L10N', function($c) {
         return $c->query('ServerContainer')->getL10N($c->query('AppName'));
      });
   }
}
```

Strings can then be translated in the following way:

```php
<?php
namespace OCA\MyApp\Service;

use \OCP\IL10N;

class AuthorService {

   private $trans;

   public function __construct(IL10N $trans){
      $this->trans = $trans;
   }

   public function getLanguageCode() {
      return $this->trans->getLanguageCode();
   }

   public sayHello() {
      return $this->trans->t('Hello');
   }

   public function getAuthorName($name) {
      return $this->trans->t('Getting author %s', array($name));
   }

   public function getAuthors($count, $city) {
      return $this->trans->n(
         '%n author is currently in the city %s',  // singular string
         '%n authors are currently in the city %s',  // plural string
         $count,
         array($city)
      );
   }
}
```

Templates

In every template the global variable $l can be used to translate the strings using its methods t() and n():

```php
<div><?php p($l->t('Showing %s files', $_['count'])); ?></div>

<button><?php p($l->t('Hide')); ?></button>
```

JavaScript

There is a global function t() available for translating strings. The first argument is your app name, the second argument is the string to translate.

```
t('myapp', 'Hello World!');
```

For advanced usage, refer to the source code core/js/l10n.js, t() is bind to OC.L10N.translate().

Hints

In case some translation strings may be translated wrongly because they have multiple meanings, you can add hints which will be shown in the Transifex web-interface:

```
<ul id="translations">
   <li id="add-new">
      <?php
         // TRANSLATORS Will be shown inside a popup and asks the user to add a
new file
         p($l->t('Add new file'));
      ?>
   </li>
</ul>
```

Creating Your Own Translatable Files

If Transifex is not the right choice or the app is not accepted for translation, generate the gettext strings by yourself by creating an l10n/ directory in the app folder and executing

```
cd /srv/http/owncloud/apps/myapp/l10n
perl l10n.pl read myapp
```

The translation script requires Locale::PO and gettext, installable via:

```
apt-get install liblocale-po-perl gettext
```

The above script generates a template that can be used to translate all strings of an app. This template is located in the folder template/ with the name myapp.pot. It can be used by your favored translation tool which then creates a .po file. The .po file needs to be placed in a folder named like the language code with the app name as filename - for example l10n/es/myapp.po. After this step the Perl script needs to be invoked to transfer the po file into our own file format that is more easily readable by the server code

```
perl l10n.pl write myapp
```

Now the following folder structure is available

```
myapp/l10n
|-- es
|   |-- myapp.po
|-- es.js
|-- es.json
|-- es.php
|-- l10n.pl
|-- templates
    |-- myapp.pot
```

You then just need the .php, .json and .js files for a working localized app.

How to automatically sync translations

1) Create an initial Transifex config within the app repository under l10n/.tx/config:

```
[main]
host = https://www.transifex.com
lang_map = ja_JP: ja

[owncloud.APP_NAME]
file_filter = <lang>/APP_NAME.po
source_file = templates/APP_NAME.pot
source_lang = en
type = PO
```

2) Give write permissions to the ownclouders user, within the ownCloud GitHub organization, just add the @owncloud/ci team with admin permissions.

3) Create a pull request at drone, just add another list item to the matrix at the bottom (the apps are sorted alphabetically).

4) After merging the pull request the translations will already be synced, afterwards it will happen every night.

Two-Factor Providers

Two-factor authentication providers apps are used to plug custom second factors into the ownCloud core. The following code was taken from the two-factor test app.

Implementing a Two-Factor Authentication Provider

Two-factor authentication providers must implement the OCP\Authentication\TwoFactorAuth\IProvider interface. The example below shows a minimalist example of such a provider.

```
<?php

namespace OCA\TwoFactor_Test\Provider;

use OCP\Authentication\TwoFactorAuth\IProvider;
use OCP\IUser;
```

```php
use OCP\Template;

class TwoFactorTestProvider implements IProvider {

    /**
     * Get unique identifier of this 2FA provider
     *
     * @return string
     */
    public function getId() {
        return 'test';
    }

    /**
     * Get the display name for selecting the 2FA provider
     *
     * @return string
     */
    public function getDisplayName() {
        return 'Test';
    }

    /**
     * Get the description for selecting the 2FA provider
     *
     * @return string
     */
    public function getDescription() {
        return 'Use a test provider';
    }

    /**
     * Get the template for rending the 2FA provider view
     *
     * @param IUser $user
     * @return Template
     */
    public function getTemplate(IUser $user) {
        // If necessary, this is also the place where you might want
        // to send out a code via e-mail or SMS.

        // 'challenge' is the name of the template
        return new Template('twofactor_test', 'challenge');
    }

    /**
     * Verify the given challenge
     *
     * @param IUser $user
     * @param string $challenge
```

```
     */
    public function verifyChallenge(IUser $user, $challenge) {
        if ($challenge === 'passme') {
            return true;
        }
        return false;
    }

    /**
     * Decides whether 2FA is enabled for the given user
     *
     * @param IUser $user
     * @return boolean
     */
    public function isTwoFactorAuthEnabledForUser(IUser $user) {
        // 2FA is enforced for all users
        return true;
    }

}
```

Registering a Two-Factor Authentication Provider

You need to inform the ownCloud core that the app provides two-factor authentication functionality. Two-factor providers are registered via info.xml.

```
<two-factor-providers>
    <provider>OCA\TwoFactor_Test\Provider\TwoFactorTestProvider</provider>
</two-factor-providers>
```

User Management

Users can be managed using the UserManager which is injected from the ServerContainer:

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Service\UserService;

class Application extends App {

   public function __construct(array $urlParams=array()){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('UserService', function($c) {
         return new UserService(
            $c->query('UserManager')
         );
      });

      $container->registerService('UserManager', function($c) {
         return $c->query('ServerContainer')->getUserManager();
      });
   }
}
```

Creating Users

Creating a user is done by passing a username and password to the create method:

```php
<?php
namespace OCA\MyApp\Service;

class UserService {

   private $userManager;

   public function __construct($userManager){
      $this->userManager = $userManager;
   }

   public function create($userId, $password) {
      return $this->userManager->create($userId, $password);
   }

}
```

Modifying Users

Users can be modified by getting a user by the userId or by a search pattern. The returned user objects can then be used to:

- Delete them
- Set a new password
- Disable/Enable them
- Get their home directory

```
<?php
namespace OCA\MyApp\Service;

class UserService {

   private $userManager;

   public function __construct($userManager){
      $this->userManager = $userManager;
   }

   public function delete($userId) {
      return $this->userManager->get($userId)->delete();
   }

   /**
    * recoveryPassword is used for the encryption app to recover the keys
    */
   public function setPassword($userId, $password, $recoveryPassword) {
      return $this->userManager->get($userId)->setPassword($password,
$recoveryPassword);
   }

   public function disable($userId) {
      return $this->userManager->get($userId)->setEnabled(false);
   }

   public function getHome($userId) {
      return $this->userManager->get($userId)->getHome();
   }
}
```

User Session Information

To login, logout or getting the currently logged in user, the UserSession has to be injected from the ServerContainer:

```php
<?php
namespace OCA\MyApp\AppInfo;

use \OCP\AppFramework\App;
use \OCA\MyApp\Service\UserService;

class Application extends App {

   public function __construct(array $urlParams=array()){
      parent::__construct('myapp', $urlParams);

      $container = $this->getContainer();

      /**
       * Controllers
       */
      $container->registerService('UserService', function($c) {
         return new UserService(
            $c->query('UserSession')
         );
      });

      $container->registerService('UserSession', function($c) {
         return $c->query('ServerContainer')->getUserSession();
      });

      // currently logged in user, userId can be gotten by calling the
      // getUID() method on it
      $container->registerService('User', function($c) {
         return $c->query('UserSession')->getUser();
      });
   }
}
```

Then users can be logged in by using:

```php
<?php
namespace OCA\MyApp\Service;

class UserService {

   private $userSession;

   public function __construct($userSession){
      $this->userSession = $userSession;
   }

   public function login($userId, $password) {
      return $this->userSession->login($userId, $password);
   }

   public function logout() {
      $this->userSession->logout();
   }

}
```

Code Signing

ownCloud supports code signing for the core releases, and for ownCloud applications. Code signing gives our users an additional layer of security by ensuring that nobody other than authorized individuals can push updates.

It also ensures that all upgrades have been executed properly, so that no files are left behind, and all old files are properly replaced. In the past, invalid updates were a significant source of errors when updating ownCloud.

FAQ

Why Did ownCloud Add Code Signing?

By supporting Code Signing we add another layer of security which ensures that nobody, other than authorized individuals, can push updates for applications. This ensures proper upgrades.

Do We Lock Down ownCloud?

The ownCloud project is open source and always will be. We do not want to make it more difficult for our users to run ownCloud. Any code signing errors on upgrades will not prevent ownCloud from running, but will display a warning on the Admin page. For applications that are not tagged Official the code signing process is optional.

Is ownCloud Not Open Source Anymore?

The ownCloud project is open source and always will be. The code signing process is optional, though highly recommended. The code check for the core parts of ownCloud is enabled when the ownCloud release version branch has been set to stable.

For custom distributions of ownCloud it is recommended to change the release version branch in version.php to something else than stable.

Is Code Signing Mandatory For Apps?

If you intend to upload your app to the Marketplace, yes, code signing *is* mandatory. If the app will only be installed directly in an ownCloud installation, then code signing is *optional*, for all third-party applications.

Technical details

ownCloud uses a X.509 based approach to handle authentication of code. Each ownCloud release contains the certificate of a shipped ownCloud Code Signing Root Authority. The private key of this certificate is only accessible to the project leader, who may grant trusted project members with a copy of this private key.

This Root Authority is only used for signing certificate signing requests (CSRs) for additional certificates. Certificates issued by the Root Authority must always to be limited to a specific scope, usually the application identifier. This enforcement is done using the CN attribute of the certificate.

Code signing is then done by creating a signature.json file with the following content:

hashes: This is an array of all files in the folder with their corresponding SHA-512 hashes.

certificate: This is the certificate used for signing.

- It has to be issued by the ownCloud Root Authority
- Its CN needs to be permitted to perform the required action.

signature: This is a signature of the hashes which can be verified using the certificate. Having the certificate bundled within the signature.json file has the advantage that even if a developer loses their certificate, future updates can still be ensured by having a new certificate issued.

How Code Signing Affects Apps in the ownCloud Marketplace

- Unsigned apps can't be uploaded to the marketplace. They can be installed manually, but the warning: "Integrity check failed", will always be visible.
- Apps which have been signed in a previous release **MUST** be code-signed in all future releases as well, otherwise the update will be refused.

How to Get Your App Signed

The following commands require that you have OpenSSL installed on your machine. Ensure that you keep all generated files to sign your application. The following examples will assume that you are trying to sign an application named **contacts**.

Firstly, generate a private key and CSR. This can be done with the following command.

```
# Replace "contacts" with your application identifier.
openssl req -nodes -newkey rsa:4096 -keyout contacts.key -out contacts.csr -subj
"/CN=contacts"
```

Then, post the CSR on https://github.com/owncloud/appstore-issues, and configure your GitHub account to show your mail address in your profile. ownCloud might ask you for further information to verify that you're the legitimate owner of the application. Make sure to keep the private key file (contacts.key) secret and not disclose it to any third-parties.

ownCloud will then provide you with the signed certificate.

Finally, run ./occ integrity:sign-app to sign your application, and specify your private and the public key as well as the path to the application. A valid example looks like:

```
./occ integrity:sign-app \
  --privateKey=/home/user/contacts.key \
  --certificate=/home/user/CA/contacts.crt \
  --path=/home/user/Programming/contacts`
```

The occ tool will store a signature.json file within the appinfo folder of your application. Then compress the application folder, naming it contacts.tar.gz, and upload it to https://marketplace.owncloud.com/. Be aware that making any changes to the application, after it has been signed, requires it to be signed again. So if you do not want to have some files shipped remove them before running the signing command.

In case you lose your certificate please submit a new CSR as described above and mention that you have lost the previous one. ownCloud will revoke the old certificate.

If you maintain an app together with multiple people it is recommended to designate a release manager responsible for the signing process as well as the uploading to marketplace. If case this is not feasible, and multiple certificates are required, ownCloud can create them on a case by case basis. We do not recommend developers to share their private key.

Errors

The following errors can be encountered when trying to verify a code signature. For information about how to get access to those results please refer to the Issues section of the ownCloud Server Administration manual.

INVALID_HASH

- The file has a different hash than specified within signature.json. This usually happens when the file has been modified after writing the signature data.

MISSING_FILE

- The file cannot be found but has been specified within signature.json. Either a required file has been left out, or signature.json needs to be edited.

EXTRA_FILE

- The file does not exist in signature.json. This usually happens when a file has been removed and signature.json has not been updated.

EXCEPTION

- Another exception has prevented the code verification. There are currently these following exceptions:
  - Signature data not found.
    - The app has mandatory code signing enforced but no signature.json file has been found in its appinfo folder.
  - Certificate is not valid.
    - The certificate has not been issued by the official ownCloud Code Signing Root Authority.

- Certificate is not valid for required scope. (Requested: %s, current: %s)
  - The certificate is not valid for the defined application. Certificates are only valid for the defined app identifier and cannot be used for others.
- Signature could not get verified.
  - There was a problem with verifying the signature of signature.json.

Tutorial

In this tutorial, you'll learn how to create an ownCloud application, by stepping through the process of creating one to manage a set of notes. The application will support *listing*, *viewing*, *creating*, *updating*, and *deleting* notes. It will step through as many concepts and techniques as possible, while not using concepts, just to do so.

Minimum Requirements

Before you can develop ownCloud applications, as with developing other software applications, you have to ensure that you have a working development environment.

To do that:

- First, ensure that your development environment meets the minimum requirements
- Then, create the core files that any ownCloud application needs

There aren't many; all that you'll need is:

- PHP, with a minimum version of 5.6, though ideally 7.1
- A copy of ownCloud core
- A working installation of ownCloud server

To find out more, read through the Development Environment section. When you've done everything that it suggests, you're ready to begin developing an ownCloud application.

The Request Life Cycle

Before we dive in to creating an application, it's important to have an overview of how the request life cycle of an ownCloud application works.

If you are not interested in the internals or don't want to execute anything before and after your controller, feel free to skip this section and continue directly with defining your app's routes.

As with other web-based applications, it's centered around an HTTP request, which typically consists of the following, four, components:

- **A URL**: e.g. /index.php/apps/myapp/something
- **Request Parameters**: e.g. ?something=true&name=tom
- **A Method**: e.g. GET
- **Request headers**: e.g. Accept: application/json

These requests are, in turn, handled by five ownCloud components:

- The Front Controller
- The Router
- Middleware

- The Dependency Injection Container
- The Controller

The Front Controller

All requests are sent to ownCloud's Front Controller: index.php, which in turn executes lib/base.php. This file:

- Inspects the HTTP headers
- Abstracts away differences between different web servers
- Initializes the core classes

Following this, ownCloud then loads its core applications; these are:

- The authentication backends
- The filesystem handler
- The logging handler

With these three applications loaded, the remaining initialization steps are then executed. These are:

- Attempt to authenticate the user is made.
- Load and execute all the remaining applications' main files. To do this, the application's main file appinfo/app.php is loaded and executed. If you want to execute code before your application is loaded, you need to place code in your app's main file.
- Load all the routes in the applications' appinfo/routes.php.
- Execute the router.

With the setup completed, ownCloud then handles the user's request.

The Router

The router:

- Parses the application's routing configuration file: appinfo/routes.php.
- Inspects the request's method and URL
- Retrieves the handling controller from the DI container.
- Passes control to the dispatcher

The dispatcher:

- Handles the requested routes by running hooks, called Middleware, before and after invoking the controller which handles the route
- Executes the controller method
- Renders the request's output

Middleware

Middleware is a convenient way to execute common tasks, such as custom authentication, before or after a controller method is executed. You can execute middleware at the following locations:

- Before calling the controller method
- After calling the controller method

- After an exception is thrown (also if it is thrown from middleware, e.g., if an authentication request fails)
- Before the output is rendered

The Dependency Injection Container

The Dependency Injection (DI) container is where you define all the services (or dependencies) that your application will need; in particular, all of your application's controllers. A key benefit of DI containers is that they handle all dependency instantiation. This means that you no longer have to rely on either globals or singletons.

The Controller

The controller contains the code that you actually want to run when a request has come in. Think of it like a callback that is executed if everything before went fine. The controller collects all the information necessary to perform the request, such as from the route and environment, and returns a response.

This response is then run through follow-up middleware (afterController and beforeOutput) for final processing. When those steps are complete, HTTP headers are then set along with the body of the response to the client.

The Core Application Files

Now that you know how the request life cycle works, let's look at the core application files. Any ownCloud application, at its most elementary, only needs a few files and directories; these are:

```
.
├── appinfo                 # Contains app metadata and configuration
│   ├── app.php
│   ├── application.php
│   ├── info.xml
│   └── routes.php
└── lib                   # Contains the application's class files
    └── Controller          # Contains the application's controllers
```

In addition to these, there are several additional, commonly used, directories:

- css/: Contains the CSS files
- js/: Contains the JavaScript files
- templates/: Contains the templates
- tests/: Contains the tests

Now let's get an understanding of the core configuration files.

appinfo/info.xml

This stores the application's properties, or metadata, and is one of the most important files. Rather like a composer.json file (only in XML format), in this file you can set details such as the application's: *id*, *name*, *description*, *license*, *author*, *version*, *namespace*, *category*, and *dependencies*.

In appinfo/info.xml, add the following XML, changing it as necessary:

```
<?xml version="1.0"?>
<info>
   <id>ownnotes</id>
   <name>Own Notes</name>
   <description>My first ownCloud App</description>
   <licence>AGPL</licence>
   <author>Your Name</author>
   <version>0.0.1</version>
   <namespace>OwnNotes</namespace>
   <category>tool</category>
   <dependencies>
      <owncloud min-version="9" />
      <owncloud max-version="10" />
   </dependencies>
</info>
```

Pay careful attention to the namespace element. This element defines the application's relative namespace. This namespace, in turn, sits inside a parent ownCloud namespace, called OCA. As the application's namespace is OwnNotes, then it's fully-qualified namespace is OCA\\OwnNotes.

To learn more about the options able to be stored in this file, check out the App Metadata section of the documentation.

appinfo/app.php

The appinfo/app.php is the first file that is loaded and executed. It usually contains the application's core configuration settings. These can include:

- **id:** This is the string under which your app will be referenced in ownCloud.
- **order:** Indicates the order in which your application will appear in the apps menu.
- **href:** The application's default route, rendered when the application's first loaded.
- **icon:** The application's icon.
- **name:** The application's title used in ownCloud.

To start off with, in appinfo/app.php, add the following code:

```
<?php

\OC::$server->getNavigationManager()->add(function () {
    $urlGenerator = \OC::$server->getURLGenerator();
    return [
        // The string under which your app will be referenced in owncloud
        'id' => 'ownnotes',

        // The sorting weight for the navigation.
        // The higher the number, the higher will it be listed in the navigation
        'order' => 10,

        // The route that will be shown on startup
        'href' => $urlGenerator->linkToRoute('ownnotes.page.index'),

        // The icon that will be shown in the navigation, located in img/
        'icon' => $urlGenerator->imagePath('ownnotes', 'ownnotes.svg'),

        // The application's title, used in the navigation & the settings page of your app
        'name' => \OC::$server->getL10N('ownnotes')->t('Test App'),
    ];
});
```

It can also contain background jobs and hook registrations, as in the example below.

```
// execute OCA\MyApp\BackgroundJob\Task::run when cron is called
\OC::$server->getJobList()->add('OCA\MyApp\BackgroundJob\Task');

// execute OCA\MyApp\Hooks\User::deleteUser before a user is being deleted
\OCP\Util::connectHook('OC_User', 'pre_deleteUser', 'OCA\MyApp\Hooks\User',
'deleteUser');
```

It is also possible to include JavaScript or CSS for other apps, by placing the addScript or addStyle functions inside this file as well. However, this is strongly discouraged, because the file is loaded on each request, as well as for requests that do not return HTML, such as JSON and WebDAV.

```
<?php

\OCP\Util::addScript('myapp', 'script');  // include js/script.js for every app
\OCP\Util::addStyle('myapp', 'style');    // include css/style.css for every app
```

lib/Controller/PageController.php

While not strictly necessary, if you want to do anything of value, you're likely going to need a controller. This can be to render page content, API content, or something else entirely. In lib/Controller/PageController.php, add the following code:

```php
<?php
namespace OCA\ownnotes\Controller;

use OCP\AppFramework\{
    Controller,
    Http\TemplateResponse
};

/**
 - Define a new page controller
 */
class PageController extends Controller {
    /**
     - @NoCSRFRequired
     */
    public function index() {
        return ['test' => 'hi'];
    }
}
```

What we're doing here is to create a minimalist controller with one action, index, which is what will handle the route that we'll define shortly. The index function returns an array, which we'll see next.

appinfo/routes.php

As the name implies, in this file you register your application's routes, and then link them to a handler. In appinfo/routes.php, add the following code:

```php
<?php

namespace OCA\ownnotes\AppInfo;

$application = new Application();
$application->registerRoutes($this, [
    'routes' => [
        [
            // The handler is the PageController's index method
            'name' => 'page#index',
            // The route
            'url' => '/',
            // Only accessible with GET requests
            'verb' => 'GET'
        ],
    ]
]);
```

appinfo/application.php

This is the core class of the application. Here, you setup your controllers among a

range of other things. In appinfo/application.php, add the following code:

```
<?php
namespace OCA\ownnotes\AppInfo;

use \OCP\AppFramework\App;
use \OCA\ownnotes\Controller\PageController;

class Application extends App {
   public function __construct(array $urlParams=array()){
      parent::__construct('ownnotes', $urlParams);

      $container = $this->getContainer();
      $container->registerService('PageController', function($c) {
         return new PageController(
            $c->query('AppName'),
            $c->query('Request')
         );
      });
   }
}
```

Create the Core File & Directory Structure

To create these, in a new directory that will be called ownnotes, run the following code in your terminal, from where you want to create the new project:

```
mkdir -p ownnotes/{appinfo,lib/Controller}
touch appinfo/{app,application,routes}.php appinfo/info.xml
lib/Controller/PageController.php
```

Routes & Controllers

Routes

A typical web application consists of both server side and client side code. The glue between those two parts are the URLs. In the case of the own notes application, the following URLs will be used:

- **GET /**: Returns the interface in HTML format
- **GET /notes**: Returns a list of all notes in JSON format
- **GET /notes/1**: Returns a note with the id 1 in JSON format
- **DELETE /notes/1**: Deletes a note with the id 1
- **POST /notes**: Creates a new note by passing in JSON format
- **PUT /notes/1**: Updates a note with the id 1 by passing in JSON format

On the client side we can call these URLs with the following jQuery code:

```
// example for calling the PUT /notes/1 URL
var baseUrl = OC.generateUrl('/apps/ownnotes');
var note = {
    title: 'New note',
    content: 'This is the note text'
};
var id = 1;
$.ajax({
    url: baseUrl + '/notes/' + id,
    type: 'PUT',
    contentType: 'application/json',
    data: JSON.stringify(note)
}).done(function (response) {
    // handle success
}).fail(function (response, code) {
    // handle failure
});
```

On the server side, we need to register a callback that is executed once the request comes in. The callback will be a method on a controller and the controller will be connected to the URL with a route .

To do that, we create the routes configuration file: ownnotes/appinfo/routes.php, which you can see the definition for below.

```
<?php
return [
    'routes' => [
        ['name' => 'page#index', 'url' => '/', 'verb' => 'GET'],
        ['name' => 'note#index', 'url' => '/notes', 'verb' => 'GET'],
        ['name' => 'note#show', 'url' => '/notes/{id}', 'verb' => 'GET'],
        ['name' => 'note#create', 'url' => '/notes', 'verb' => 'POST'],
        ['name' => 'note#update', 'url' => '/notes/{id}', 'verb' => 'PUT'],
        ['name' => 'note#destroy', 'url' => '/notes/{id}', 'verb' => 'DELETE']
    ]
];
```

A handy feature of routing in ownCloud is that as the final five routes are so similar, they can be abbreviated by adding a resource instead:

```
<?php
return [
   'resources' => [
      'note' => ['url' => '/notes']
   ],
   'routes' => [
      ['name' => 'page#index', 'url' => '/', 'verb' => 'GET']
   ]
];
```

Let's look at the route below first, so that you get a better understanding of how they're composed.

```
<?php
return ['routes' => [
   ['name' => 'page#index', 'url' => '/', 'verb' => 'GET']
]];
```

This route (/) is accessible only via a GET request and is called page#index. When called, the request will be handled by OCA\\OwnNotes\\PageController's index method. The reason why is defined in the route's name. The name is composed of the name of a controller and a method on that controller, separated by a hash symbol.

Controllers

The controller, more specifically the controller function, as in other MVC-based frameworks, is the central place of logic for a route (or action). These functions, as you would expect, can return a range of responses to the user, including: JSON, HTML, XML, and plain text; a redirect or 404 Not Found response, or the download of a file.

In the example below, we'll return an HTML response, based on the contents of a template file, using the TemplateResponse object. The TemplateResponse object renders a template located in an application's templates directory.

```
<?php
namespace OCA\OwnNotes\Controller;

use OCP\IRequest;
use OCP\AppFramework\Http\TemplateResponse;
use OCP\AppFramework\Controller;

class PageController extends Controller {

   public function __construct($AppName, IRequest $request){
      parent::__construct($AppName, $request);
   }

   /**
    * @NoAdminRequired
    * @NoCSRFRequired
    */
   public function index() {
      // Renders ownnotes/templates/main.php
      return new TemplateResponse('ownnotes', 'main');
   }

}
```

The first argument to the constructor specifies which application's template directory to search. The second argument specifies the template to use, minus file extension (.php). Templates are, effectively, not much more than the original PHP files, which were a combination of PHP and HTML.

The OCP namespace maps to ownCloud/core/lib/public.

The @NoAdminRequired and @NoCSRFRequired annotations in index's docblock above turn off security checks, as they're not necessary for this method. See Controllers for more information.

With an initial overview of controllers (and templates) completed, we'll now create the core of a controller which handles AJAX requests for the application. Create a new controller, called ownnotes/lib/Controller/NoteController.php, with the following content:

```
<?php
namespace OCA\OwnNotes\Controller;

use OCP\IRequest;
use OCP\AppFramework\Controller;

class NoteController extends Controller {

   public function __construct($AppName, IRequest $request){
      parent::__construct($AppName, $request);
   }
```

```
    /**
     * @NoAdminRequired
     */
    public function index() {
        // empty for now
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function show($id) {
        // empty for now
    }

    /**
     * @NoAdminRequired
     *
     * @param string $title
     * @param string $content
     */
    public function create($title, $content) {
        // empty for now
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     * @param string $title
     * @param string $content
     */
    public function update($id, $title, $content) {
        // empty for now
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function destroy($id) {
        // empty for now
    }

}
```

You can see that it's largely the same as the PageController, but with a range of CRUD

methods. Take special note of show, create, update, and destroy. The parameters to these functions are extracted from the request body and the URL, using the controller method's variable names.

We're not going to do anything further in this chapter. However, we'll flesh out the controller in the next chapter on database interaction.

Database Connectivity

The Database Schema

Now that the application's routes and two controllers have been setup and wired together, we'll flesh out NotesController so that the notes can be saved in the database. But to do that, we first need to create the database schema by creating ownnotes/appinfo/database.xml, with the following content:

```
<database>
  <name>*dbname*</name>
  <create>true</create>
  <overwrite>false</overwrite>
  <charset>utf8</charset>
  <table>
    <name>*dbprefix*ownnotes_notes</name>
    <declaration>
      <field>
        <name>id</name>
        <type>integer</type>
        <notnull>true</notnull>
        <autoincrement>true</autoincrement>
        <unsigned>true</unsigned>
        <primary>true</primary>
        <length>8</length>
      </field>
      <field>
        <name>title</name>
        <type>text</type>
        <length>200</length>
        <default></default>
        <notnull>true</notnull>
      </field>
      <field>
        <name>user_id</name>
        <type>text</type>
        <length>200</length>
        <default></default>
        <notnull>true</notnull>
      </field>
      <field>
        <name>content</name>
        <type>clob</type>
        <default></default>
        <notnull>true</notnull>
      </field>
    </declaration>
  </table>
</database>
```

The schema consists of one table: ownnotes_notes, which has four fields:

- **id:** An integer
- **title:** A text field
- **user_id:** A text field
- **content:** A CLOB field

With the file created, the version tag in ownnotes/appinfo/info.xml needs to be

increased. This causes ownCloud to trigger the update process when you next load (or reload) the ownCloud UI. Part of the update process includes run database migrations, which will create the database table defined in the migration above.

Data Entities

Now that the tables are created, we want to map the database search results to a PHP object. That way, we're able to manage the data more precisely. To do that, create an entity in new file, called: ownnotes/lib/Db/Note.php:

```
<?php
namespace OCA\OwnNotes\Db;

use JsonSerializable;
use OCP\AppFramework\Db\Entity;

class Note extends Entity implements JsonSerializable {

    protected $title;
    protected $content;
    protected $userId;

    public function jsonSerialize() {
        return [
            'id' => $this->id,
            'title' => $this->title,
            'content' => $this->content
        ];
    }

}
```

The id field exists in the Entity

We also define a jsonSerializable method and implement the interface, so that we're able to transform the entity to JSON, making it easy to persist and cache the information.

Data Mappers

Entities are returned from so-called data mappers. Data mappers are:

"" A layer of Mappers (473) that moves data between objects and a database while keeping them independent of each other and the mapper itself. ""

Let's create one in ownnotes/lib/Db/NoteMapper.php and add a find and findAll method:

```php
<?php
namespace OCA\OwnNotes\Db;

use OCP\IDb;
use OCP\AppFramework\Db\Mapper;

class NoteMapper extends Mapper {

   public function __construct(IDb $db) {
      parent::__construct($db, 'ownnotes_notes', '\OCA\OwnNotes\Db\Note');
   }

   public function find($id, $userId) {
      $sql = 'SELECT * FROM *PREFIX*ownnotes_notes WHERE id = ? AND user_id =
?';
      return $this->findEntity($sql, [$id, $userId]);
   }

   public function findAll($userId) {
      $sql = 'SELECT * FROM *PREFIX*ownnotes_notes WHERE user_id = ?';
      return $this->findEntities($sql, [$userId]);
   }

}
```

The first parent constructor parameter is the database connection object (or database handle), the second one is the database table and the third is the entity which the result should be mapped onto. Insert, delete and update methods are already implemented.

Connecting Databases & Controllers

Now the mapper is finished and can be passed into the controller. You can do so by adding it as a type-hinted parameter. ownCloud will figure out how to assemble them by itself.

Additionally we want to know the userId of the currently logged in user. To do so, add a $UserId parameter to the constructor, which is case-sensitive. Open ownnotes/lib/Controller/NoteController.php and change it to the following:

```
<?php
 namespace OCA\OwnNotes\Controller;

 use Exception;

 use OCP\IRequest;
 use OCP\AppFramework\Http;
 use OCP\AppFramework\Http\DataResponse;
 use OCP\AppFramework\Controller;

 use OCA\OwnNotes\Db\Note;
 use OCA\OwnNotes\Db\NoteMapper;

 class NoteController extends Controller {

    private $mapper;
    private $userId;

    public function __construct($AppName, IRequest $request, NoteMapper
$mapper, $UserId){
        parent::__construct($AppName, $request);
        $this->mapper = $mapper;
        $this->userId = $UserId;
    }

 }
```

With the constructor defined, we now need to flesh out the rest of the methods, which we previously didn't define bodies for. In index, below, we'll return a DataResponse object, which contains the result of using the Data Mapper's findAll method.

This method, which is supplied with the current user's id, retrieves all notes created by that user. A DataResponse object is used to return generic data responses. It provides a more generic response than JSONResponse, which also works with JSON data.

```
/**
 * @NoAdminRequired
 */
public function index() {
    return new DataResponse($this->mapper->findAll($this->userId));
}
```

Next, we'll flesh out the show function. This function will retrieve and return the details for a specific note. It does so by using the data mapper's find method, which is supplied with the note's and user's ids. If the note cannot be retrieved, then a DataResponse is returned, which results in a 404 Not Found response.

```
/**
 * @NoAdminRequired
 *
 * @param int $id
 */
public function show($id) {
    try {
        return new DataResponse($this->mapper->find($id, $this->userId));
    } catch(Exception $e) {
        return new DataResponse([], Http::STATUS_NOT_FOUND);
    }
}
```

Next, we'll flesh out the create method, so that we can create notes. This method receives the note's title and content from the route and sets them, along with the current user's id, on a new Note entity object. The function returns the result of calling the data mapper's insert method, which attempts to persist the Note entity in the database.

```
/**
 * @NoAdminRequired
 *
 * @param string $title
 * @param string $content
 */
public function create($title, $content) {
    $note = new Note();
    $note->setTitle($title);
    $note->setContent($content);
    $note->setUserId($this->userId);

    return new DataResponse($this->mapper->insert($note));
}
```

Next we'll flesh out the update function, which updates an existing note. Similar to the create method, it receives the note's id, title, and content from the route. It then attempts to retrieve the note, and throws an exception if it's unable to do so. If it can retrieve it, it then updates the title and content, and returns the response from calling the data mapper's update function.

```
/**
 * @NoAdminRequired
 *
 * @param int $id
 * @param string $title
 * @param string $content
 */
public function update($id, $title, $content) {
    try {
        $note = $this->mapper->find($id, $this->userId);
    } catch(Exception $e) {
        return new DataResponse([], Http::STATUS_NOT_FOUND);
    }
    $note->setTitle($title);
    $note->setContent($content);
    return new DataResponse($this->mapper->update($note));
}
```

Finally, we'll flesh out the destroy function, which deletes an existing note. This, like update, will first attempt to retrieve a note, based on the supplied id, and throw an exception if it's not able to be found. If it's able to be found, it will then be passed to the data mapper's delete function, which will delete the note from the database.

```
/**
 * @NoAdminRequired
 *
 * @param int $id
 */
public function destroy($id) {
    try {
        $note = $this->mapper->find($id, $this->userId);
    } catch(Exception $e) {
        return new DataResponse([], Http::STATUS_NOT_FOUND);
    }
    $this->mapper->delete($note);
    return new DataResponse($note);
}
```

This is all that is needed on the server side. Now let's progress to the client side.

Decoupling Controllers and Increasing Reusability

Let's now say that our app is now on the ownCloud Marketplace, and we get a request that we should save the files in the filesystem which requires access to the filesystem.

The filesystem API is quite different from the database API and throws different exceptions, which means we need to rewrite everything in the NoteController class to use it.

This is bad, because a controller's only responsibility should be to deal with incoming HTTP requests and return HTTP responses. If we need to change the controller

because the data storage was changed the code is probably too tightly coupled. So we need to add another layer in between, a layer called Service.

Let's take the logic that was inside the controller and put it into a separate class inside ownnotes/lib/Service/NoteService.php:

```
<?php
namespace OCA\OwnNotes\Service;

use Exception;
use OCP\AppFramework\Db\DoesNotExistException;
use OCP\AppFramework\Db\MultipleObjectsReturnedException;
use OCA\OwnNotes\Db\Note;
use OCA\OwnNotes\Db\NoteMapper;

class NoteService {

    private $mapper;

    public function __construct(NoteMapper $mapper){
        $this->mapper = $mapper;
    }

    public function findAll($userId) {
        return $this->mapper->findAll($userId);
    }

    private function handleException ($e) {
        if ($e instanceof DoesNotExistException ||
            $e instanceof MultipleObjectsReturnedException) {
            throw new NotFoundException($e->getMessage());
        } else {
            throw $e;
        }
    }

    public function find($id, $userId) {
        try {
            return $this->mapper->find($id, $userId);

        // In order to be able to plug in different storage backends like files
        // for instance it is a good idea to turn storage related exceptions
        // into service related exceptions so controllers and service users
        // have to deal with only one type of exception
        } catch(Exception $e) {
            $this->handleException($e);
        }
    }

    public function create($title, $content, $userId) {
        $note = new Note();
```

```
        $note->setTitle($title);
        $note->setContent($content);
        $note->setUserId($userId);
        return $this->mapper->insert($note);
    }

    public function update($id, $title, $content, $userId) {
        try {
            $note = $this->mapper->find($id, $userId);
            $note->setTitle($title);
            $note->setContent($content);
            return $this->mapper->update($note);
        } catch(Exception $e) {
            $this->handleException($e);
        }
    }

    public function delete($id, $userId) {
        try {
            $note = $this->mapper->find($id, $userId);
            $this->mapper->delete($note);
            return $note;
        } catch(Exception $e) {
            $this->handleException($e);
        }
    }

}
```

Following that, create an exception class in ownnotes/lib/Service/ServiceException.php:

```
<?php
namespace OCA\OwnNotes\Service;

use Exception;

class ServiceException extends Exception {}
```

Then, create another one in ownnotes/lib/Service/NotFoundException.php:

```
<?php
namespace OCA\OwnNotes\Service;

class NotFoundException extends ServiceException {}
```

Remember how we had all those ugly try/catch blocks that where checking for DoesNotExistException and simply returned a 404 response? Let's also refactor these into a reusable class.

Specifically, we'll use a trait, so that we can inherit methods without having to create a large inheritance hierarchy. This will be important later on when you've got controllers that inherit from the ApiController class instead. The trait is created in ownnotes/lib/Controller/Errors.php:

```
<?php

namespace OCA\OwnNotes\Controller;

use Closure;
use OCP\AppFramework\Http;
use OCP\AppFramework\Http\DataResponse;
use OCA\OwnNotes\Service\NotFoundException;

trait Errors {

   protected function handleNotFound (Closure $callback) {
      try {
         return new DataResponse($callback());
      } catch(NotFoundException $e) {
         $message = ['message' => $e->getMessage()];
         return new DataResponse($message, Http::STATUS_NOT_FOUND);
      }
   }

}
```

Now we can wire up the trait and the service inside the NoteController:

```
<?php
namespace OCA\OwnNotes\Controller;

use OCP\IRequest;
use OCP\AppFramework\Http\DataResponse;
use OCP\AppFramework\Controller;
use OCA\OwnNotes\Service\NoteService;

class NoteController extends Controller {

   private $service;
   private $userId;

   use Errors;

   public function __construct($AppName, IRequest $request,
                               NoteService $service, $UserId){
      parent::__construct($AppName, $request);
      $this->service = $service;
      $this->userId = $UserId;
   }
```

```php
    /**
     * @NoAdminRequired
     */
    public function index() {
        return new DataResponse($this->service->findAll($this->userId));
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function show($id) {
        return $this->handleNotFound(function () use ($id) {
            return $this->service->find($id, $this->userId);
        });
    }

    /**
     * @NoAdminRequired
     *
     * @param string $title
     * @param string $content
     */
    public function create($title, $content) {
        return $this->service->create($title, $content, $this->userId);
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     * @param string $title
     * @param string $content
     */
    public function update($id, $title, $content) {
        return $this->handleNotFound(function () use ($id, $title, $content) {
            return $this->service->update($id, $title, $content, $this->userId);
        });
    }

    /**
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function destroy($id) {
        return $this->handleNotFound(function () use ($id) {
            return $this->service->delete($id, $this->userId);
```

```
        });
    }

}
```

As a result of these changes, the only reason that the controller needs to be changed is when request/response related things change.

Creating Template Content

As noted in the controllers section of the tutorial, templates are, effectively, not much more than the original PHP files, which were a combination of PHP and HTML. However, they can also contain conditional logic, as you can see in the example below.

This template, in ownnotes/templates/part.content.php,contains the core form elements for creating notes. $l→t() is used to make your strings translatable and p() is used to print escaped HTML.

```
<script id="content-tpl" type="text/x-handlebars-template">
    {{#if note}}
        <div class="input"><textarea>{{ note.content }}</textarea></div>
        <div class="save"><button><?php p($l->t('Save')); ?></button></div>
    {{else}}
        <div class="input"><textarea disabled></textarea></div>
        <div class="save"><button disabled><?php p($l->t('Save'));
?></button></div>
    {{/if}}
</script>
<div id="editor"></div>
```

Creating a Navigation Menu

A navigation menu is, effectively, another template, In our example, we'll create it in ownnotes/templates/part.navigation.php.

ownCloud defines many handy CSS styles which we are going to reuse to style the navigation. Adjust the file to contain only the following code:

```
<!-- translation strings -->
<div style="display:none" id="new-note-string"><?php p($l->t('New note'));
?></div>

<script id="navigation-tpl" type="text/x-handlebars-template">
   <li id="new-note"><a href="#"><?php p($l->t('Add note')); ?></a></li>
   {{#each notes}}
      <li class="note with-menu {{#if active}}active{{/if}}"  data-id="{{ id }}">
         <a href="#">{{ title }}</a>
         <div class="app-navigation-entry-utils">
            <ul>
               <li class="app-navigation-entry-utils-menu-button svg"><button
></button></li>
            </ul>
         </div>

         <div class="app-navigation-entry-menu">
            <ul>
               <li><button class="delete icon-delete svg" title="delete"
></button></li>
            </ul>
         </div>
      </li>
   {{/each}}
</script>

<ul></ul>
```

Add JavaScript and CSS

To create a modern web application you need to write JavaScript and CSS.

JavaScript

You can use any JavaScript framework but for this tutorial we want to keep it as simple as possible and therefore only include the templating library handlebarsjs. Download the file into ownnotes/js/handlebars.js and include it at the very top of ownnotes/templates/main.php before the other scripts and styles:

```
<?php
script('ownnotes', 'handlebars');
```

The script method's first parameter specifies the application which the JavaScript should be included for. This helps increase performance by not including the JavaScript unnecessarily. The script's second parameter is the name of the JavaScript file, located in the application's js directory, minus the .js extension. In the case above, ownnotes/js/handlebars.js would be loaded.

jQuery is included by default on every page.

CSS

To include CSS, use the template's style method, as in the example below. As with script, the first parameter is the application to find the CSS file in and the second parameter is the name of the CSS file, minus the .css file extension.

```
style('ownnotes, 'style');  // adds ownnotes/css/style.css
```

ownCloud doesn't provide automatic JavaScript or CSS minification

Wiring It Up

When the page is loaded, we want all the existing notes to load. Furthermore:

- We want to display the current note when you click on it in the navigation
- A note should be deleted when we click the deleted button
- Clicking on **New note** should create a new note.

To do that open ownnotes/js/script.js and replace the example code with the following:

```
(function (OC, window, $, undefined) {
'use strict';

$(document).ready(function () {

var translations = {
    newNote: $('#new-note-string').text()
};

// this notes object holds all our notes
var Notes = function (baseUrl) {
    this._baseUrl = baseUrl;
    this._notes = [];
    this._activeNote = undefined;
};

Notes.prototype = {
    load: function (id) {
        var self = this;
        this._notes.forEach(function (note) {
            if (note.id === id) {
                note.active = true;
                self._activeNote = note;
            } else {
                note.active = false;
            }
        });
    },
    getActive: function () {
        return this._activeNote;
    },
```

```
    removeActive: function () {
        var index;
        var deferred = $.Deferred();
        var id = this._activeNote.id;
        this._notes.forEach(function (note, counter) {
            if (note.id === id) {
                index = counter;
            }
        });

        if (index !== undefined) {
            // delete cached active note if necessary
            if (this._activeNote === this._notes[index]) {
                delete this._activeNote;
            }

            this._notes.splice(index, 1);

            $.ajax({
                url: this._baseUrl + '/' + id,
                method: 'DELETE'
            }).done(function () {
                deferred.resolve();
            }).fail(function () {
                deferred.reject();
            });
        } else {
            deferred.reject();
        }
        return deferred.promise();
    },
    create: function (note) {
        var deferred = $.Deferred();
        var self = this;
        $.ajax({
            url: this._baseUrl,
            method: 'POST',
            contentType: 'application/json',
            data: JSON.stringify(note)
        }).done(function (note) {
            self._notes.push(note);
            self._activeNote = note;
            self.load(note.id);
            deferred.resolve();
        }).fail(function () {
            deferred.reject();
        });
        return deferred.promise();
    },
    getAll: function () {
```

```
        return this._notes;
    },
    loadAll: function () {
        var deferred = $.Deferred();
        var self = this;
        $.get(this._baseUrl).done(function (notes) {
            self._activeNote = undefined;
            self._notes = notes;
            deferred.resolve();
        }).fail(function () {
            deferred.reject();
        });
        return deferred.promise();
    },
    updateActive: function (title, content) {
        var note = this.getActive();
        note.title = title;
        note.content = content;

        return $.ajax({
            url: this._baseUrl + '/' + note.id,
            method: 'PUT',
            contentType: 'application/json',
            data: JSON.stringify(note)
        });
    }
};

// this will be the view that is used to update the html
var View = function (notes) {
    this._notes = notes;
};

View.prototype = {
    renderContent: function () {
        var source = $('#content-tpl').html();
        var template = Handlebars.compile(source);
        var html = template({note: this._notes.getActive()});

        $('#editor').html(html);

        // handle saves
        var textarea = $('#app-content textarea');
        var self = this;
        $('#app-content button').click(function () {
            var content = textarea.val();
            var title = content.split('\n')[0]; // first line is the title

            self._notes.updateActive(title, content).done(function () {
                self.render();
```

```
        }).fail(function () {
            alert('Could not update note, not found');
        });
    });
  },
  renderNavigation: function () {
    var source = $('#navigation-tpl').html();
    var template = Handlebars.compile(source);
    var html = template({notes: this._notes.getAll()});

    $('#app-navigation ul').html(html);

    // create a new note
    var self = this;
    $('#new-note').click(function () {
        var note = {
            title: translations.newNote,
            content: ''
        };

        self._notes.create(note).done(function() {
            self.render();
            $('#editor textarea').focus();
        }).fail(function () {
            alert('Could not create note');
        });
    });

    // show app menu
    $('#app-navigation .app-navigation-entry-utils-menu-button').click(function ()
{
        var entry = $(this).closest('.note');
        entry.find('.app-navigation-entry-menu').toggleClass('open');
    });

    // delete a note
    $('#app-navigation .note .delete').click(function () {
        var entry = $(this).closest('.note');
        entry.find('.app-navigation-entry-menu').removeClass('open');

        self._notes.removeActive().done(function () {
            self.render();
        }).fail(function () {
            alert('Could not delete note, not found');
        });
    });

    // load a note
    $('#app-navigation .note > a').click(function () {
        var id = parseInt($(this).parent().data('id'), 10);
```

```
            self._notes.load(id);
            self.render();
            $('#editor textarea').focus();
        });
    },
    render: function () {
        this.renderNavigation();
        this.renderContent();
    }
};

var notes = new Notes(OC.generateUrl('/apps/ownnotes/notes'));
var view = new View(notes);
notes.loadAll().done(function () {
    view.render();
}).fail(function () {
    alert('Could not load notes');
});


});

})(OC, window, jQuery);
```

Apply Finishing Touches

Now, the only thing left is to style the textarea in a nicer fashion. To do that open ownnotes/css/style.css and replace the content with the following CSS code:

```
#app-content-wrapper {
    height: 100%;
}

#editor {
    height: 100%;
    width: 100%;
}

#editor .input {
    height: calc(100% - 51px);
    width: 100%;
}

#editor .save {
    height: 50px;
    width: 100%;
    text-align: center;
    border-top: 1px solid #ccc;
    background-color: #fafafa;
}

#editor textarea {
    height: 100%;
    width: 100%;
    border: 0;
    margin: 0;
    border-radius: 0;
    overflow-y: auto;
}

#editor button {
    height: 44px;
}
```

Congratulations! You've written your first ownCloud app. You can now either try to further improve the tutorial notes app or start writing your own app.

Add a RESTful API (optional)

A RESTful API allows other apps such as Android or iPhone apps to access and change your notes. Since syncing is a big core component of ownCloud it is a good idea to add, and document, your own RESTful API.

Because we put our logic into the NoteService class it is very easy to reuse it. The only pieces that need to be changed are the annotations which disable the CSRF check (not needed for a REST call usually) and add support for CORS so your API can be accessed from other webapps.

With that in mind create a new controller in ownnotes/lib/Controller/NoteApiController.php:

```php
<?php
namespace OCA\OwnNotes\Controller;

use OCP\IRequest;
use OCP\AppFramework\Http\DataResponse;
use OCP\AppFramework\ApiController;

use OCA\OwnNotes\Service\NoteService;

class NoteApiController extends ApiController {

    private $service;
    private $userId;

    use Errors;

    public function __construct($AppName, IRequest $request,
                                NoteService $service, $UserId){
        parent::__construct($AppName, $request);
        $this->service = $service;
        $this->userId = $UserId;
    }

    /**
     * @CORS
     * @NoCSRFRequired
     * @NoAdminRequired
     */
    public function index() {
        return new DataResponse($this->service->findAll($this->userId));
    }

    /**
     * @CORS
     * @NoCSRFRequired
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function show($id) {
        return $this->handleNotFound(function () use ($id) {
            return $this->service->find($id, $this->userId);
        });
    }

    /**
     * @CORS
     * @NoCSRFRequired
     * @NoAdminRequired
     *
```

```
     * @param string $title
     * @param string $content
     */
    public function create($title, $content) {
        return $this->service->create($title, $content, $this->userId);
    }

    /**
     * @CORS
     * @NoCSRFRequired
     * @NoAdminRequired
     *
     * @param int $id
     * @param string $title
     * @param string $content
     */
    public function update($id, $title, $content) {
        return $this->handleNotFound(function () use ($id, $title, $content) {
            return $this->service->update($id, $title, $content, $this->userId);
        });
    }

    /**
     * @CORS
     * @NoCSRFRequired
     * @NoAdminRequired
     *
     * @param int $id
     */
    public function destroy($id) {
        return $this->handleNotFound(function () use ($id) {
            return $this->service->delete($id, $this->userId);
        });
    }

}
```

All that is left is to connect the controller to a route and enable the built in pre-flighted CORS method which is defined in the ApiController base class:

```
<?php
return [
   'resources' => [
      'note' => ['url' => '/notes'],
      'note_api' => ['url' => '/api/0.1/notes']
   ],
   'routes' => [
      ['name' => 'page#index', 'url' => '/', 'verb' => 'GET'],
      ['name' => 'note_api#preflighted_cors', 'url' => '/api/0.1/{path}',
       'verb' => 'OPTIONS', 'requirements' => ['path' => '.+']]
   ]
];
```

It is a good idea to version your API in your URL

Testing the API

You can test the API by running a GET request with **curl**:

```
curl -u user:password http://localhost:{std-port-
http}/index.php/apps/ownnotes/api/0.1/notes
```

Since the NoteApiController is basically identical to the NoteController, the unit test for it simply inherits its tests from the NoteControllerTest. Create the file ownnotes/tests/Unit/Controller/NoteApiControllerTest.php:

```
<?php
namespace OCA\OwnNotes\Tests\Unit\Controller;

require_once __DIR__ . '/NoteControllerTest.php';

class NoteApiControllerTest extends NoteControllerTest {

   public function setUp() {
      parent::setUp();
      $this->controller = new NoteApiController(
         'ownnotes', $this->request, $this->service, $this->userId
      );
   }

}
```

Writing Tests

Tests are essential for having happy users and a carefree life. No one wants their users to rant about your app breaking their ownCloud or being buggy. To do that you need to test your app. Since this amounts to a ton of repetitive tasks, we need to automate the tests.

Unit Tests

A unit test is a test that tests a class in isolation. It is very fast and catches most of the bugs, so we want many unit tests. Because ownCloud uses Dependency Injection to assemble your app, it is very easy to write unit tests by passing mocks into the constructor. A simple test for the update method can be added by adding this to ownnotes/tests/Unit/Controller/NoteControllerTest.php:

```
<?php
namespace OCA\OwnNotes\Tests\Unit\Controller;

use PHPUnit_Framework_TestCase;

use OCP\AppFramework\Http;
use OCP\AppFramework\Http\DataResponse;

use OCA\OwnNotes\Service\NotFoundException;


class NoteControllerTest extends PHPUnit_Framework_TestCase {

    protected $controller;
    protected $service;
    protected $userId = 'john';
    protected $request;

    public function setUp() {
        $this->request = $this->getMockBuilder('OCP\IRequest')->getMock();
        $this->service = $this->getMockBuilder('OCA\OwnNotes\Service\NoteService')
            ->disableOriginalConstructor()
            ->getMock();
        $this->controller = new NoteController(
            'ownnotes', $this->request, $this->service, $this->userId
        );
    }

    public function testUpdate() {
        $note = 'just check if this value is returned correctly';
        $this->service->expects($this->once())
            ->method('update')
            ->with($this->equalTo(3),
                   $this->equalTo('title'),
                   $this->equalTo('content'),
                  $this->equalTo($this->userId))
            ->will($this->returnValue($note));

        $result = $this->controller->update(3, 'title', 'content');

        $this->assertEquals($note, $result->getData());
    }
```

```
    public function testUpdateNotFound() {
        // test the correct status code if no note is found
        $this->service->expects($this->once())
            ->method('update')
            ->will($this->throwException(new NotFoundException()));

        $result = $this->controller->update(3, 'title', 'content');

        $this->assertEquals(Http::STATUS_NOT_FOUND, $result->getStatus());
    }

}
```

We can and should also create a test for the NoteService class:

```
<?php
namespace OCA\OwnNotes\Tests\Unit\Service;

use PHPUnit_Framework_TestCase;

use OCP\AppFramework\Db\DoesNotExistException;

use OCA\OwnNotes\Db\Note;

class NoteServiceTest extends PHPUnit_Framework_TestCase {

    private $service;
    private $mapper;
    private $userId = 'john';

    public function setUp() {
        $this->mapper = $this->getMockBuilder('OCA\OwnNotes\Db\NoteMapper')
            ->disableOriginalConstructor()
            ->getMock();
        $this->service = new NoteService($this->mapper);
    }

    public function testUpdate() {
        // the existing note
        $note = Note::fromRow([
            'id' => 3,
            'title' => 'yo',
            'content' => 'nope'
        ]);
        $this->mapper->expects($this->once())
            ->method('find')
            ->with($this->equalTo(3))
            ->will($this->returnValue($note));
```

```
        // the note when updated
        $updatedNote = Note::fromRow(['id' => 3]);
        $updatedNote->setTitle('title');
        $updatedNote->setContent('content');
        $this->mapper->expects($this->once())
            ->method('update')
            ->with($this->equalTo($updatedNote))
            ->will($this->returnValue($updatedNote));

        $result = $this->service->update(3, 'title', 'content', $this->userId);

        $this->assertEquals($updatedNote, $result);
    }


    /**
     * @expectedException OCA\OwnNotes\Service\NotFoundException
     */
    public function testUpdateNotFound() {
        // test the correct status code if no note is found
        $this->mapper->expects($this->once())
            ->method('find')
            ->with($this->equalTo(3))
            ->will($this->throwException(new DoesNotExistException(`)));

        $this->service->update(3, 'title', 'content', $this->userId);
    }

}
```

If PHPUnit is installed we can run the tests inside ownnotes/ with the following command:

```
phpunit
```

You need to adjust the ownnotes/tests/Unit/Controller/PageControllerTest file to get the tests passing: remove the testEcho method since that method is no longer present in your PageController and do not test the user id parameters since they are not passed anymore

Integration Tests

Integration tests are slow and need a fully working instance but make sure that our classes work well together. Instead of mocking out all classes and parameters we can decide whether to use full instances or replace certain classes. Because they are slow we don't want as many integration tests as unit tests.

In our case we want to create an integration test for the udpate method without mocking out the NoteMapper class so we actually write to the existing database. To do that create a new file called ownnotes/tests/Integration/NoteIntegrationTest.php with the following content:

```php
<?php
namespace OCA\OwnNotes\Tests\Integration\Controller;

use OCP\AppFramework\Http\DataResponse;
use OCP\AppFramework\App;
use Test\TestCase;

use OCA\OwnNotes\Db\Note;

class NoteIntregrationTest extends TestCase {

    private $controller;
    private $mapper;
    private $userId = 'john';

    public function setUp() {
        parent::setUp();
        $app = new App('ownnotes');
        $container = $app->getContainer();

        // only replace the user id
        $container->registerService('UserId', function($c) {
            return $this->userId;
        });

        $this->controller = $container->query(
            'OCA\OwnNotes\Controller\NoteController'
        );

        $this->mapper = $container->query(
            'OCA\OwnNotes\Db\NoteMapper'
        );
    }

    public function testUpdate() {
        // create a new note that should be updated
        $note = new Note();
        $note->setTitle('old_title');
        $note->setContent('old_content');
        $note->setUserId($this->userId);

        $id = $this->mapper->insert($note)->getId();

        // fromRow does not set the fields as updated
        $updatedNote = Note::fromRow([
            'id' => $id,
            'user_id' => $this->userId
        ]);
        $updatedNote->setContent('content');
        $updatedNote->setTitle('title');
```

```
        $result = $this->controller->update($id, 'title', 'content');

        $this->assertEquals($updatedNote, $result->getData());

        // clean up
        $this->mapper->delete($result->getData());
    }

}
```

To run the integration tests change into the ownnotes directory and run

```
phpunit -c phpunit.integration.xml
```

Mobile Development

In this section, you will find the core information that you need to develop mobile apps that work with ownCloud.

Android Application Development

ownCloud provides an official ownCloud Android client, which gives its users access to their files on their ownCloud. It also includes functionality like automatically uploading pictures and videos to ownCloud. For third party application developers, ownCloud offers the ownCloud Android library under the MIT license.

Android ownCloud Client development

If you are interested in working on the ownCloud android client, you can find the source code in github. The setup and process of contribution is documented here. You might want to start with doing one or two junior jobs to get into the code and note our General Contributor Guidelines. Note that contribution to the Android client require signing the ownCloud Contributor Agreement.

ownCloud Android Library

This document will describe how to the use ownCloud Android Library. The ownCloud Android Library allows a developer to communicate with any ownCloud server; among the features included are file synchronization, upload and download of files, delete rename files and folders, etc.

This library may be added to a project and seamlessly integrates any application with ownCloud. The tool needed is any IDE for Android. This guide includes some screenshots showing examples in Eclipse.

Library Installation

Obtaining the library

The ownCloud Android library may be obtained from the following GitHub repository:

https://github.com/owncloud/android-library

Once obtained, this code should be compiled. The Github repository not only contains the library, but also a sample project, sample_client sample_client

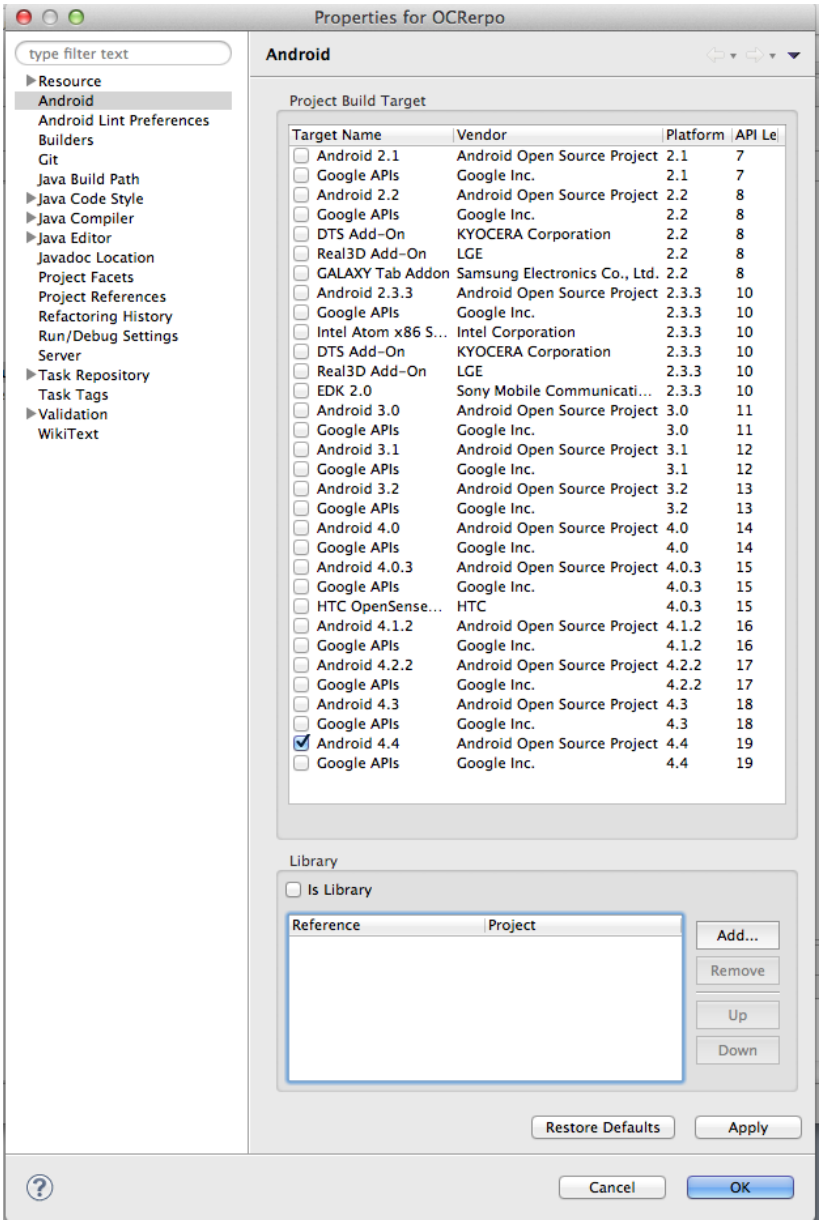
properties/android/librerias, which will assist in learning how to use the library.

Add the library to a project

There are different methods to add an external library to a project, then we will describe one of them.

1. Compile the ownCloud Android Library
2. Define a dependency within your project.

For that, access to Properties > Android > Library ** and click on add and select the ownCloud Android library



Then all the public classes and methods of the library will be available for your own app.

Examples

Init the library

Start using the library; it is needed to init the object mClient that will be in charge of keeping the communication with the server.

Code example

```
public class MainActivity extends Activity
                    implements  OnRemoteOperationListener,
                                OnDatatransferProgressListener {
private OwnCloudClient mClient;
private Handler mHandler = new Handler();

...

public void onCreate(Bundle savedInstanceState) {

...

// Parse URI to the base URL of the ownCloud server
Uri serverUri = Uri.parse(getString(R.string.server_base_url));

// Create client object to perform remote operations
mClient = OwnCloudClientFactory.createOwnCloudClient(
        serverUri,
        this,
        // Activity or Service context
        true);
```

Set credentials

Authentication on the app is possible by 3 different methods:

- Basic authentication, user name and password
- Bearer access token (oAuth2)
- Cookie (SAML-based single-sign-on)

Code example

```
package com.owncloud.android.lib.common;

public class OwnCloudClient extends HttpClient {
  ...
  // Set basic credentials
  client.setCredentials(
     OwnCloudCredentialsFactory.newBasicCredentials(username, password)
  );
  // Set bearer access token
  client.setCredentials(
     OwnCloudCredentialsFactory.newBearerCredentials(accessToken)
  );
  // Set SAML2 session token
  client.setCredentials(
     OwnCloudCredentialsFactory.newSamlSsoCredentials(cookie)
  );
}
```

Create a folder

Create a new folder on the cloud server, the info needed to be sent is the path of the new folder.

Code example

_ _

```
private void startFolderCreation(String newFolderPath) {
  CreateRemoteFolderOperation createOperation = new
CreateRemoteFolderOperation(newFolderPath, false);
  createOperation.execute( mClient , this , mHandler);
}

@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof CreateRemoteFolderOperation) {
    if (result.isSuccess()) {
    // do your stuff here
    }
  }
  ...
}
```

Read folder

Get the content of an existing folder on the cloud server, the info needed to be sent is the path of the folder, in the example shown it has been asked the content of the root folder. As answer of this method, it will be received an array with all the files and folders stored in the selected folder.

Code example

```
private void startReadRootFolder() {
  ReadRemoteFolderOperation refreshOperation = new
ReadRemoteFolderOperation(FileUtils.PATH_SEPARATOR);
  // root folder
  refreshOperation.execute(mClient, this, mHandler);
}


@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof ReadRemoteFolderOperation) {
    if (result.isSuccess()) {
      List< RemoteFile > files = result.getData();
      // do your stuff here
    }
  }
  ...
}
```

Read file

Get information related to a certain file or folder, information obtained is: filePath, filename, isDirectory, size and date.

Code example

```
private void startReadFileProperties(String filePath) {
  ReadRemoteFileOperation readOperation = new
ReadRemoteFileOperation(filePath);
  readOperation.execute(mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof ReadRemoteFileOperation) {
    if (result.isSuccess()) {
      RemoteFile file = result.getData()[0];
      // do your stuff here
    }
  }
  ...
}
```

Delete file or folder

Delete a file or folder on the cloud server. The info needed is the path of folder/file to be deleted.

Code example

```
private void startRemoveFile(String filePath) {
  RemoveRemoteFileOperation removeOperation = new
RemoveRemoteFileOperation(remotePath);
  removeOperation.execute( mClient , this , mHandler);
}

@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof RemoveRemoteFileOperation) {
    if (result.isSuccess()) {
      // do your stuff here
    }
  }
  ...
}
```

Download a file

Download an existing file on the cloud server. The info needed is path of the file on the server and targetDirectory, path where the file will be stored on the device.

Code example

```
private void startDownload(String filePath, File targetDirectory) {
  DownloadRemoteFileOperation downloadOperation = new
DownloadRemoteFileOperation(filePath, targetDirectory.getAbsolutePath());
  downloadOperation.addDatatransferProgressListener(this);
  downloadOperation.execute( mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish( RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof DownloadRemoteFileOperation) {
    if (result.isSuccess()) {
      // do your stuff here
    }
  }
}

@Override
public void onTransferProgress( long progressRate, long totalTransferredSoFar,
long totalToTransfer, String fileName) {
mHandler.post( new Runnable() {
  @Override
  public void run() {
    // do your UI updates about progress here
  }
});
}
```

Upload a file

Upload a new file to the cloud server. The info needed is fileToUpload, path where the file is stored on the device, remotePath, path where the file will be stored on the server and mimeType.

Code example

```
private void startUpload (File fileToUpload, String remotePath, String
mimeType) {
 UploadRemoteFileOperation uploadOperation = new
UploadRemoteFileOperation( fileToUpload.getAbsolutePath(), remotePath,
mimeType);
 uploadOperation.addDatatransferProgressListener(this);
 uploadOperation.execute(mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
 if (operation instanceof UploadRemoteFileOperation) {
  if (result.isSuccess()) {
   // do your stuff here
  }
 }
}

@Override
public void onTransferProgress(long progressRate, long totalTransferredSoFar,
long totalToTransfer, String fileName) {
 mHandler.post( new Runnable() {
  @Override
  public void run() {
   // do your UI updates about progress here
  }
 });
}
```

Move a file or folder

Move an exisintg file or folder to a different location in the ownCloud server. Parameters needed are the path to the file or folder to move, and the new path desired for it. The parent folder of the new path must exist in the server.

When the parameter `overwrite' is set to `true', the file or folder is moved even if the new path is already used by a different file or folder. This one will be replaced by the former.

Code example

```
private void startFileMove(String filePath, String newFilePath, boolean
overwrite) {
 MoveRemoteFileOperation moveOperation = new
MoveRemoteFileOperation(filePath, newFilePath, overwrite);
 moveOperation.execute( mClient , this , mHandler);
}

@Override
public void onRemoteOperationFinish(RemoteOperation operation,
RemoteOperationResult result) {
 if (operation instanceof MoveRemoteFileOperation) {
  if (result.isSuccess()) {
  // do your stuff here
  }
 }
 ...
}
```

Read shared items by link

Get information about what files and folder are shared by link (the object mClient contains the information about the server url and account)

Code example

```
private void startAllSharesRetrieval() {
 GetRemoteSharesOperation getSharesOp = new
GetRemoteSharesOperation();
 getSharesOp.execute( mClient , this , mHandler);
}

@Override
public void onRemoteOperationFinish( RemoteOperation operation,
RemoteOperationResult result) {
 if (operation instanceof GetRemoteSharesOperation) {
  if (result.isSuccess()) {
   ArrayList< OCShare > shares = new ArrayList< OCShare >();
   for (Object obj: result.getData()) {
    shares.add(( OCShare) obj);
   }
   // do your stuff here
  }
 }
}
```

Get the share resources for a given file or folder

Get information about what files and folder are shared by link on a certain folder. The info needed is filePath, path of the file/folder on the server, the Boolean variable, getReshares, come from the Sharing api, from the moment it is not in use within the

ownCloud Android library.

Code example

```
private void startSharesRetrievalForFileOrFolder(String filePath, boolean
getReshares) {
 GeteRemoteSharesForFileOperation operation = new
GetRemoteSharesForFileOperation(filePath, getReshares, false);
 operation.execute( mClient, this, mHandler);
}

private void startSharesRetrievalForFilesInFolder(String folderPath, boolean
getReshares) {
 GetRemoteSharesForFileOperation operation = new
GetRemoteSharesForFileOperation(folderPath, getReshares, true);
 operation.execute( mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish( RemoteOperation operation,
RemoteOperationResult result) {
 if (operation instanceof GetRemoteSharesForFileOperation) {
  if (result.isSuccess()) {
   ArrayList< OCShare > shares = new ArrayList< OCShare >();
   for (Object obj: result.getData()) {
    shares.add(( OCShare) obj);
   }
   // do your stuff here
  }
}
}
```

Share link of file or folder

Share a file or a folder from your cloud server by link.

The info needed is filePath, the path of the item that you want to share and Password, this comes from the Sharing api, from the moment it is not in use within the ownCloud Android library.

Code example

```
private void startCreationOfPublicShareForFile(String filePath, String
password) {
 CreateRemoteShareOperation operation = new
CreateRemoteShareOperation(filePath, ShareType.PUBLIC_LINK, "", false,
password, 1);
 operation.execute( mClient , this , mHandler);
}

private void startCreationOfGroupShareForFile(String filePath, String
groupId) {
 CreateRemoteShareOperation operation = new
CreateRemoteShareOperation(filePath, ShareType.GROUP, groupId, false , "",
31);
 operation.execute(mClient, this, mHandler);
}

private void startCreationOfUserShareForFile(String filePath, String userId) {
 CreateRemoteShareOperation operation = new
CreateRemoteShareOperation(filePath, ShareType.USER, userId, false, "", 31);
 operation.execute(mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish( RemoteOperation operation,
RemoteOperationResult result) {
 if (operation instanceof CreateRemoteShareOperation) {
  if (result.isSuccess()) {
   OCShare share = (OCShare) result.getData ().get(0);
   // do your stuff here
  }
 }
}
```

Delete a share resource

Stop sharing by link a file or a folder from your cloud server.

The info needed is the object OCShare that you want to stop sharing by link.

Code example

```
private void startShareRemoval(OCShare share) {
  RemoveRemoteShareOperation operation = new
RemoveRemoteShareOperation((int) share.getIdRemoteShared());
  operation.execute( mClient, this, mHandler);
}

@Override
public void onRemoteOperationFinish( RemoteOperation operation,
RemoteOperationResult result) {
  if (operation instanceof RemoveRemoteShareOperation) {
    if (result.isSuccess()) {
    // do your stuff here
    }
  }
}
```

Tips

- Credentials must be set before calling any method
- Paths must not be on URL Encoding
- Correct path: https://example.com/owncloud/remote.php/dav/PopMusic
- Wrong path: https://example.com/owncloud/remote.php/dav/Pop%20Music/
- There are some forbidden characters to be used in folder and files names on the server, same on the ownCloud Android Library: /,<,>,:,",\`,?,*.
- Upload and download actions may be cancelled thanks to the objects uploadOperation.cancel(), downloadOperation.cancel()
- Unit tests, before launching unit tests you have to enter your account information (server url, user and password) on TestActivity.java.

iOS Application Development

ownCloud provides an official ownCloud iOS client, which gives its users access to their files on their ownCloud. It also includes functionality like automatically uploading pictures and videos to ownCloud.

For third party application developers, ownCloud offers the ownCloud iOS library under the MIT license.

iOS ownCloud Client development

If you are interested in working on the ownCloud iOS client, you can find the source code in github. The setup and process of contribution is documented here.

You might want to start with doing one or two junior jobs to get into the code and note our General Contributor Guidelines.

Note that contribution to the iOS client require signing the iOS addendum to the ownCloud Contributor Agreement. You are permitted to test the iOS client on Apple hardware thanks to the iOS license exception.

ownCloud iOS Library

This document will describe how to the use ownCloud iOS library. The ownCloud iOS library for iOS allows a developer to communicate with any ownCloud server; among the features included are file synchronization, upload and download of files, delete rename and move of files and folders and share files or folders by link among others.

This library may be added to a project and seamlessly integrates any application with ownCloud.

The tool needed is Xcode 6, this guide includes some screenshots showing examples in Xcode 6.

Library Installation

Obtaining the library

The ownCloud iOS library may be obtained from the following Github repository:

git@github.com:owncloud/ios-library.git

Once obtained, this code should be compiled with Xcode 6. The Github repository not only contains the library, ownCloud iOS library, but also contains a sample project, OCLibraryExample, which will assist in learning how to use the library.

Add the library to a project

There are two methods to add this library to a project.

- Reference the headers and library binary file (.a) directly.
- Include the library as a subproject.

Which method to choose depends on user preference as well as whether the source code and project file of the static library are available.

Reference headers and library binary files

Follow these steps if this is the desired method.

1. Compile the ownCloud iOS library and run the project. A libownCloudiOS.a file will be generated.

The following files are required:

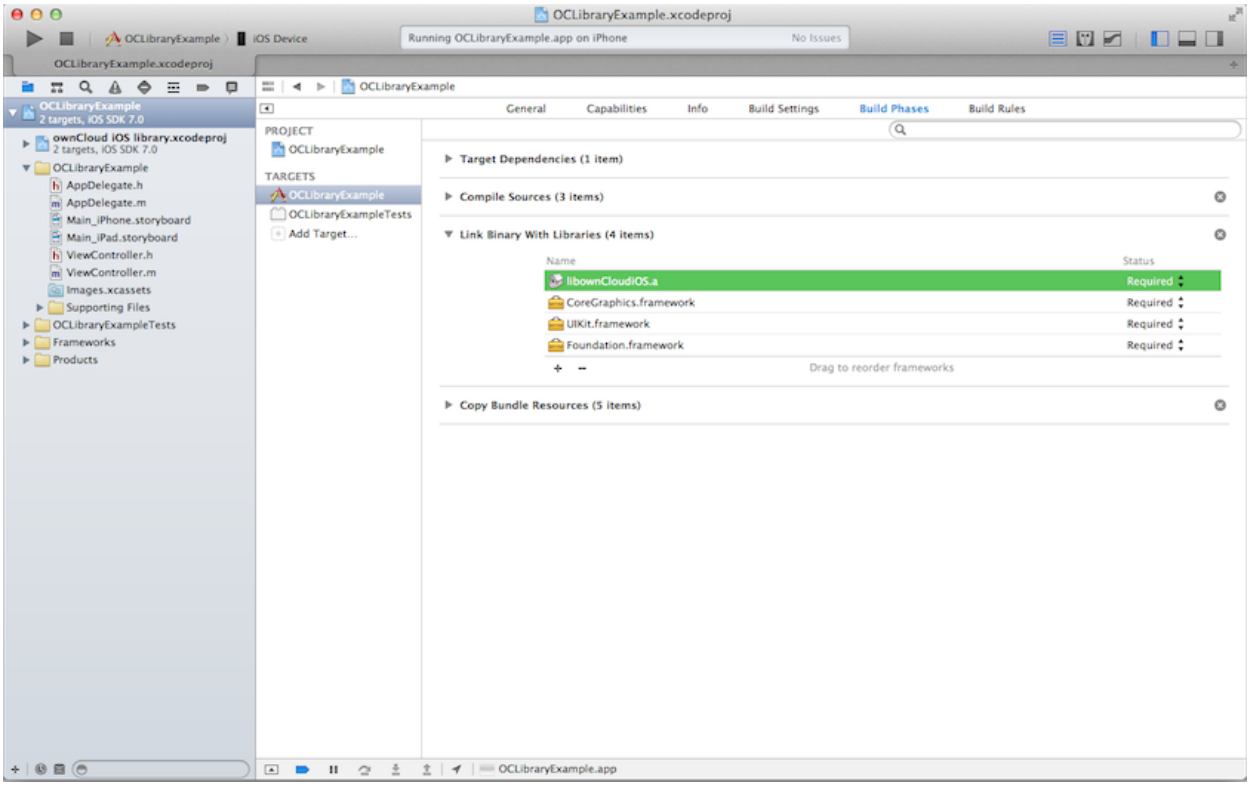
Library file

- libownCloudiOS.a (Library)

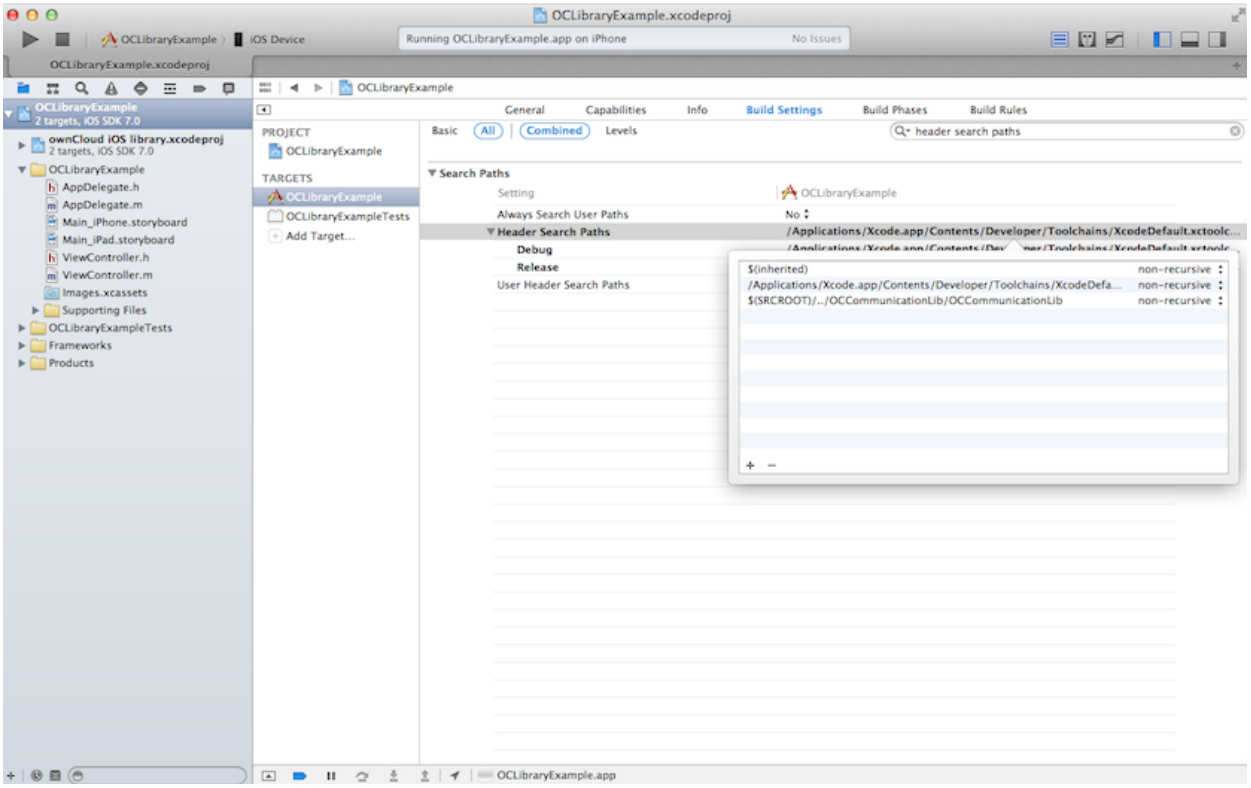
Library Classes

- OCCommunication.h (Accessors) Import in the communication class
- OCErrorMsg.h (Error Messages) Import in the communication class
- OCFileDto.h and OCFileDto.m (File/Folder object) Import when using
- readFolder and readFile methods
- OCFrameworkConstants.h (Customize constants)

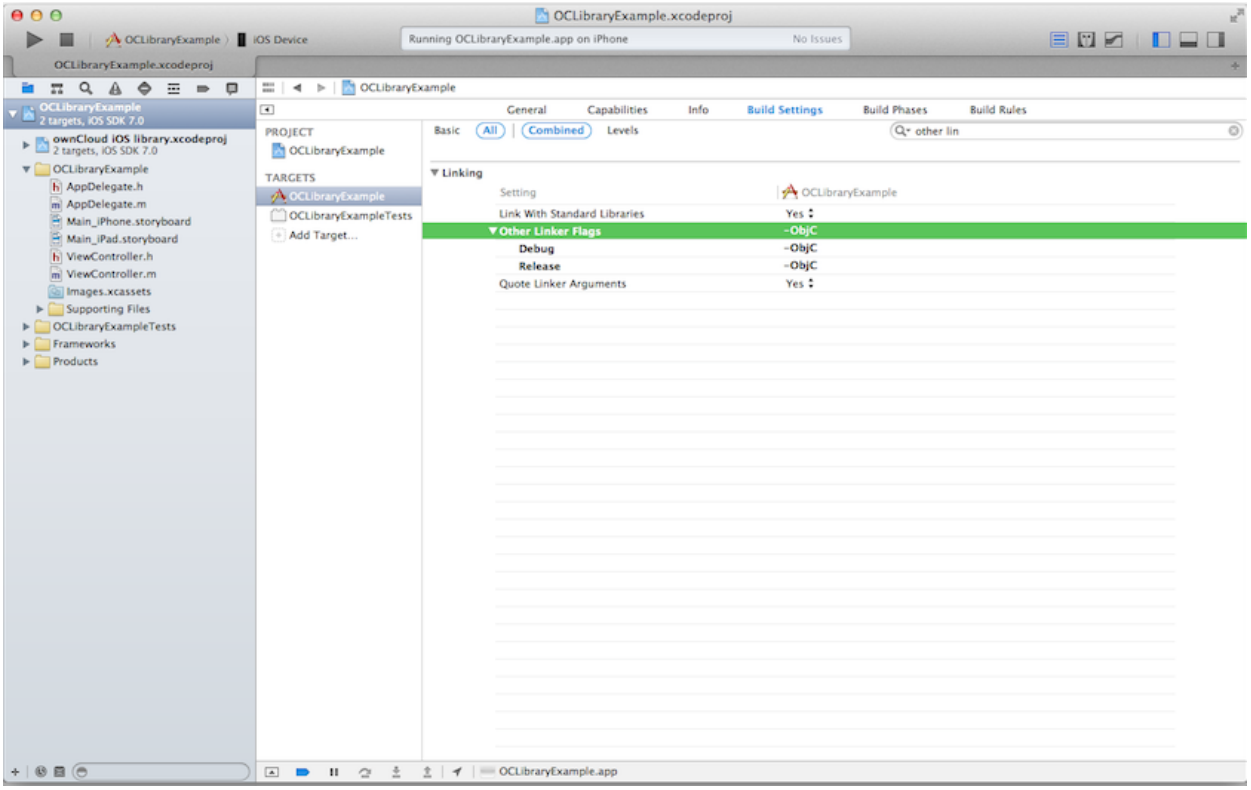
2. Add the library file to the project. From the Build Phases tab, scroll to Link binary files and select the + to add a library. Select the library file.



3. Add the path of the library header files. Under the Build Settings tab, select the target library and add the path in the Header Search Paths field.



4. Remaining in the Build Setting tab, add the flag -Obj-C under the Other Linker Flags option.

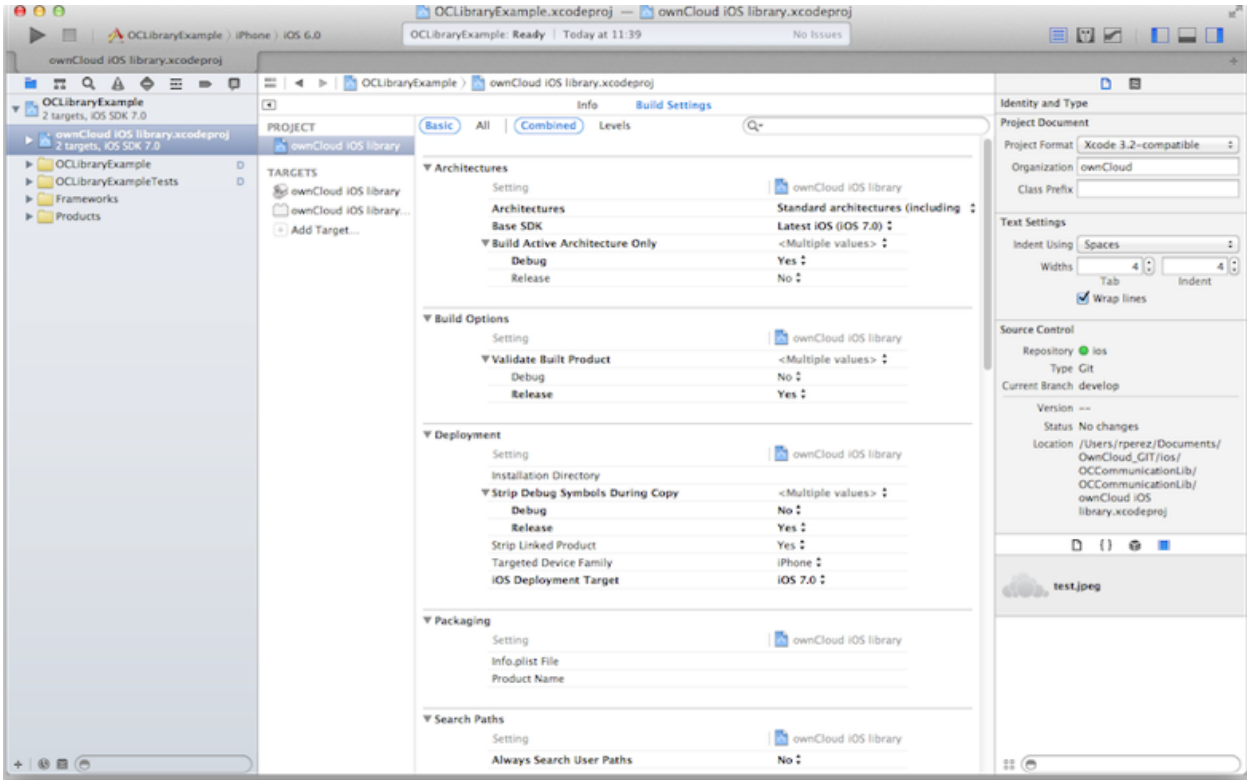


At this stage, the library is included on your project and you can start communicating with the ownCloud server.

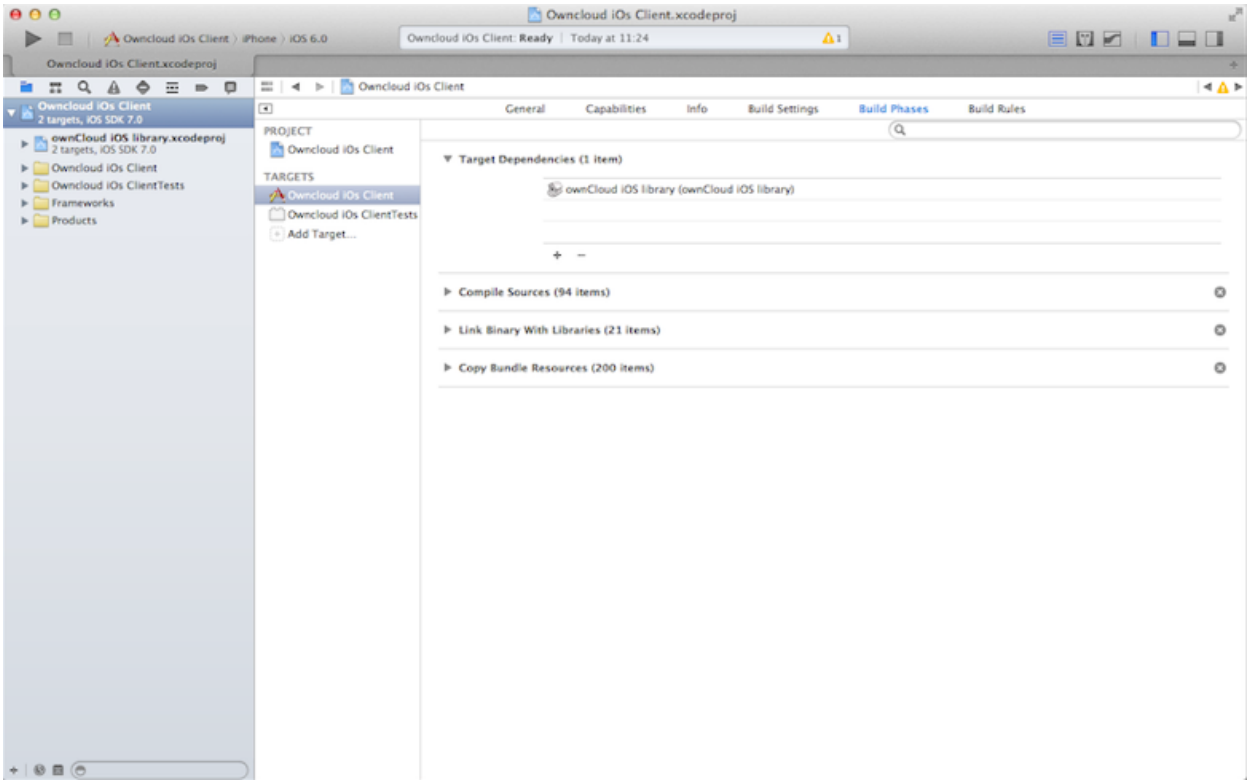
Include the library as a subproject

Follow these steps if this is the desired method.

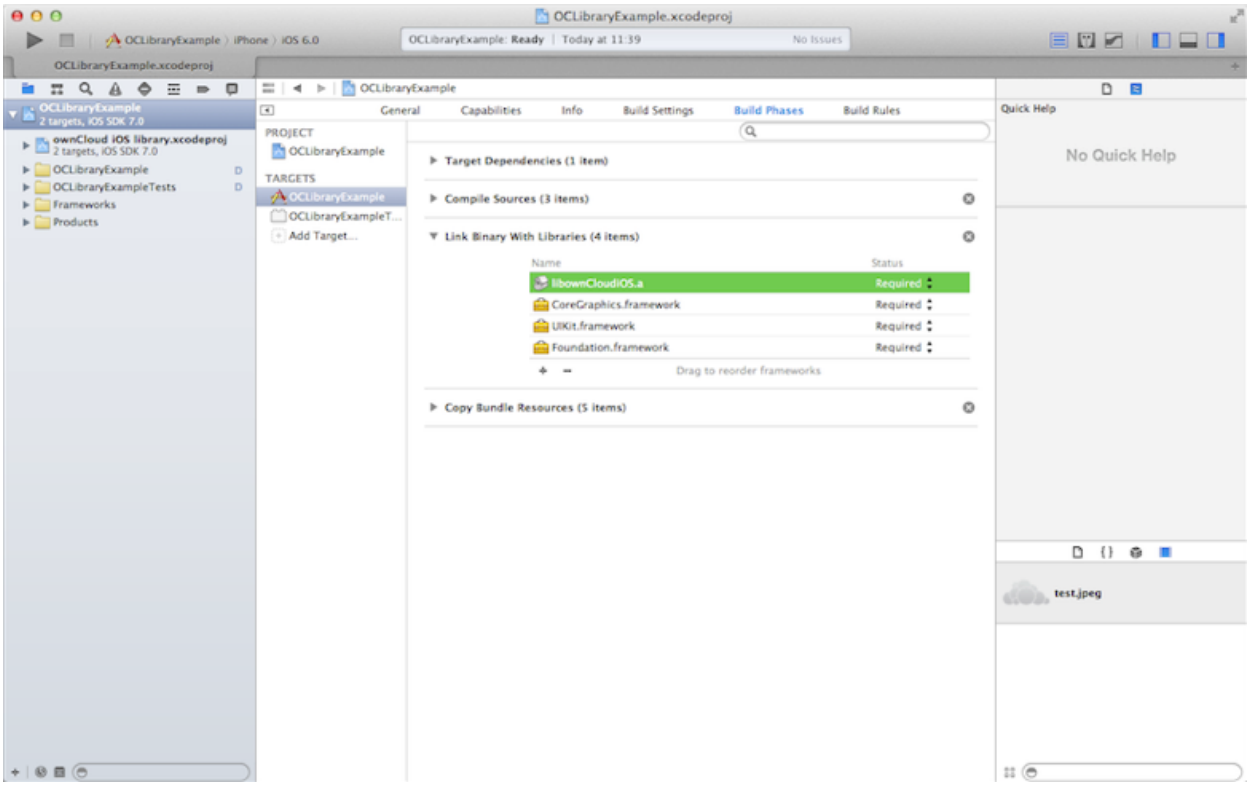
5. Add the file ownCloud iOS library.xcodeproj to the project via drag and drop.



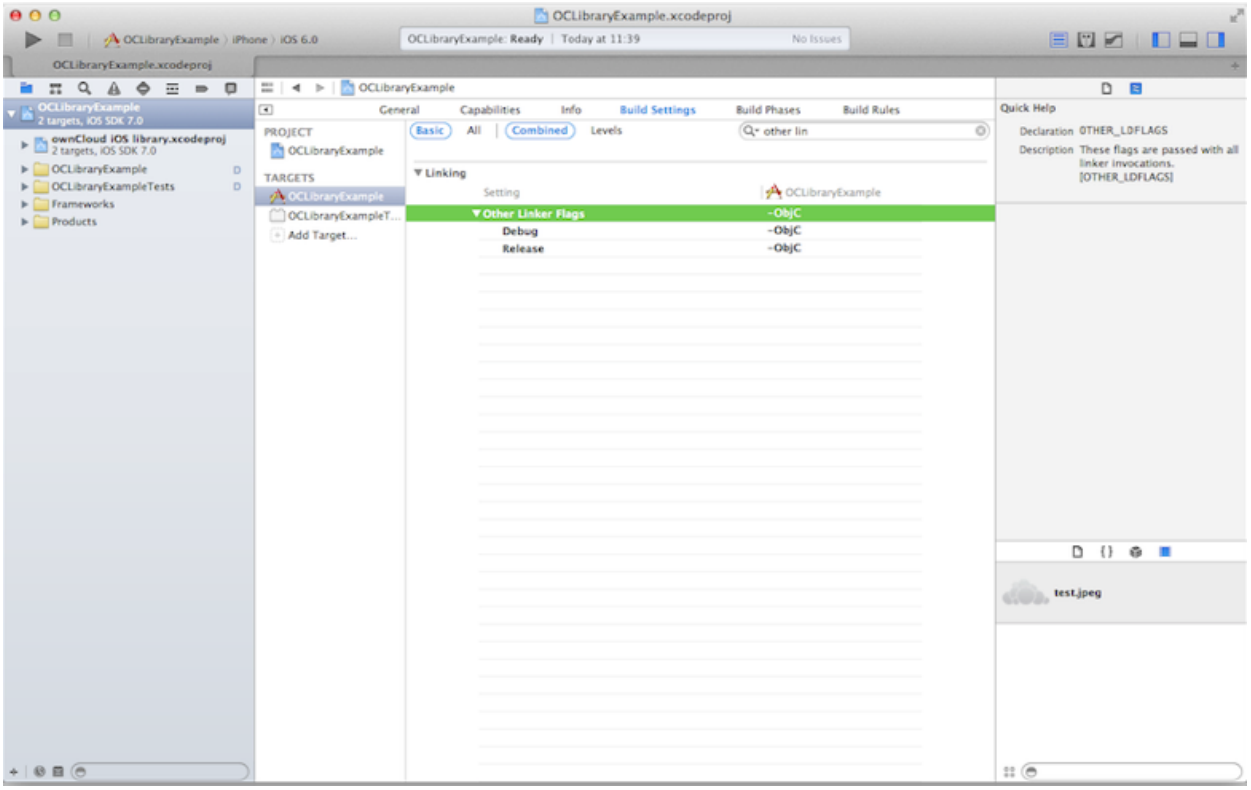
6. Within the project, navigate to the Build Phases tab. Under the Target Dependencies section, select the `+' and choose the library target.



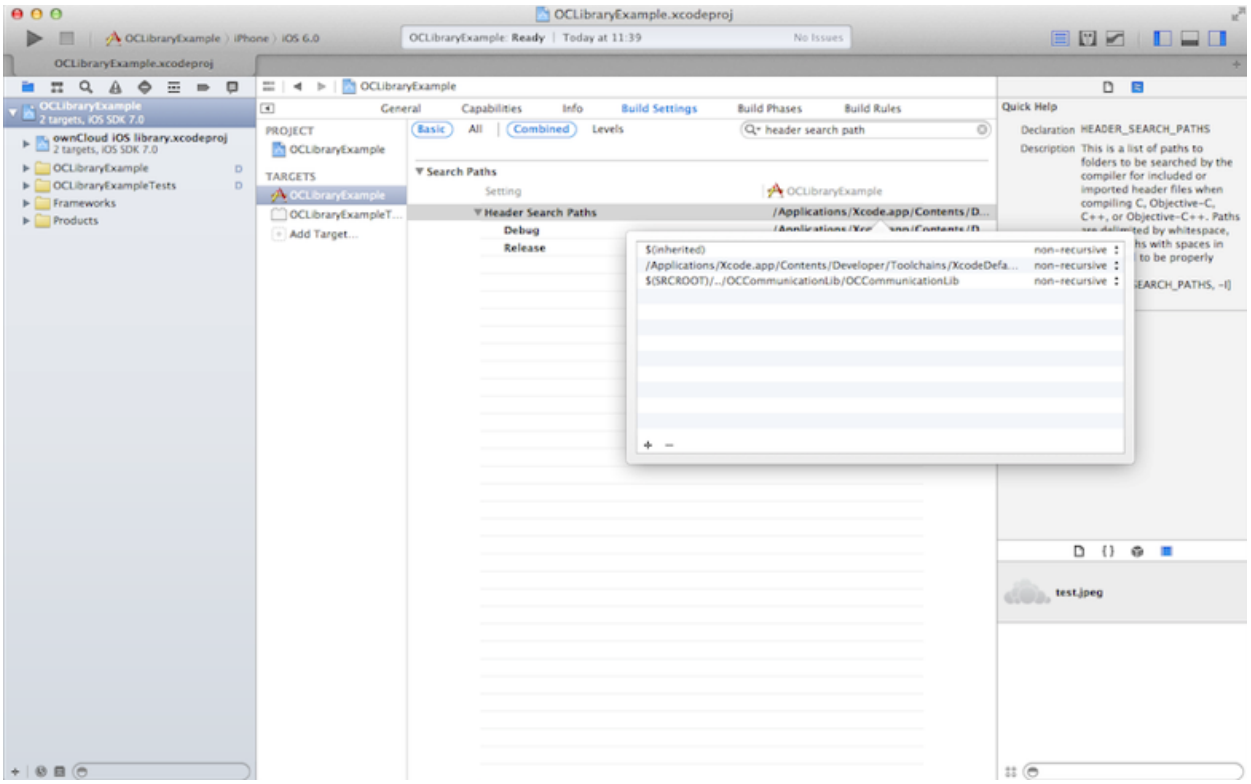
7. Link the library file to the project target. Under the Build Phases tab, select the +' under the `Link Binary with Libraries section and select the library file.



8. Add the flag -Obj-C to Other Linker Flags under the project target on the Build Settings tab.



9. Finally add the path of the library headers. Under the Build Settings tab, add the path under the Header Search Paths option.



Sources

- Creating a static library in iOS tutorial (raywenderlich.com)
- Creating Static Library in iOS App Development

Examples

Init the library

Start using the library, it is needed to init the object OCCommunication.

We recommend using the singleton method in the AppDelegate class in order to use the ownCloud iOS library.

Code example

```
#import "OCCommunication.h"

+ (OCCommunication *)sharedOCCommunication
{
 static OCCommunication* sharedOCCommunication = nil;

 if (sharedOCCommunication == nil)
 {
  sharedOCCommunication = [ [ OCCommunicationalloc] init ];
 }

 return sharedOCCommunication;
}
```

Also could happen that you need to overwrite the class AFURLSessionManager to manage SSL Certificates

```
#import "OCCommunication.h"

+ (OCCommunication*)sharedOCCommunication
{
static OCCommunication* sharedOCCommunication = nil;
if (sharedOCCommunication == nil)
{
//Network Upload queue for NSURLSession (iOS 7)
    NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration
backgroundSessionConfiguration:k_session_name];
    configuration.HTTPMaximumConnectionsPerHost = 1;
    configuration.requestCachePolicy =
NSURLRequestReloadIgnoringLocalCacheData;
    OCURLSessionManager *uploadSessionManager = [[OCURLSessionManager alloc]
initWithSessionConfiguration:configuration];
    [uploadSessionManager.operationQueue setMaxConcurrentOperationCount:1];
    [uploadSessionManager
setSessionDidReceiveAuthenticationChallengeBlock:^NSURLSessionAuthChallenge
Disposition (NSURLSession *session, NSURLAuthenticationChallenge *challenge,
NSURLCredential * __autoreleasing *credential) {
        return NSURLSessionAuthChallengePerformDefaultHandling;
    }];

    sharedOCCommunication = [[OCCommunication alloc]
initWithUploadSessionManager:uploadSessionManager];

}
return sharedOCCommunication;
}
```

Set credentials

Authentication on the app is possible by 3 different methods:

- Basic authentication, user name and password
- Cookie
- Token (oAuth)

Code example

```
#Basic authentication, user name and password
[[ AppDelegate sharedOCCommunication ] setCredentialsWithUser : userName
andPassword : password ];

#Authentication with cookie
[[ AppDelegate sharedOCCommunication ] setCredentialsWithCookie : cookie ];

#Authentication with token
[[ AppDelegate sharedOCCommunication ] setCredentialsOauthWithToken : token ];
```

Create a folder

Create a new folder on the cloud server, the info needed to be sent is the path of the new folder.

Code example

```
[[ AppDelegate sharedOCCommunication ] createFolder :path onCommunication : [
AppDelegate sharedOCCommunication ]

successRequest :^( NSHTTPURLResponse *response, NSString *redirectedServer) {
//Folder Created
}

failureRequest :^( NSHTTPURLResponse *response, NSError *error) {

//Failure

switch (response.statusCode) {

case kOCErrorServerUnauthorized :
  //Bad credentials
  break;
case kOCErrorServerForbidden :
  //Forbidden
  break;
case kOCErrorServerPathNotFound :
  //Not Found
  break;
case kOCErrorServerTimeout :
  //timeout
  break;
default:
  //default
  break;
}

}
errorBeforeRequest :^( NSError *error) {
//Error before request

if (error.code == OCErrorForbidenCharacters) {
  //Forbidden characters
}
else
{
  //Other error
}

}];
```

Read folder

Get the content of an existing folder on the cloud server, the info needed to be sent is the path of the folder. As answer of this method, it will be received an array with all the files and folders stored in the selected folder.

Code example

```
[[ AppDelegate sharedOCCommunication] readFolder:path onCommunication:[
AppDelegate sharedOCCommunication]

successRequest:^( NSHTTPURLResponse *response, NSArray *items, NSString
*redirectedServer) {
  //Success
  for ( OCFileDto * ocFileDto in items) {
    NSLog( @"item path: %@%@" , ocFileDto.filePath, ocFileDto.fileName);
  }
}

failureRequest:^( NSHTTPURLResponse *response, NSError *error) {

//Failure
switch (response.statusCode) {
case kOCErrorServerPathNotFound :
  //Path not found
  break;
case kOCErrorServerUnauthorized :
  //Bad credentials
  break;
case kOCErrorServerForbidden :
  //Forbidden
  break;
case kOCErrorServerTimeout :
  //Timeout
  break ;
default :
  break;
}

}];
```

Read file

Get information related to a certain file or folder. Although, more information can be obtained, the library only gets the eTag.

Other properties of the file or folder may be obtained: filePath, filename, isDirectory, size and date

Code example

```
[[ AppDelegate sharedOCCommunication ] readFile :path onCommunication :[
AppDelegate sharedOCCommunication ]

successRequest :^( NSHTTPURLResponse *response, NSArray *items, NSString
*redirectedServer) {

OCFileDto *ocFileDto = [items objectAtIndex : 0 ];
NSLog ( @"item etag: %lld" , ocFileDto.  etag); }
failureRequest :^( NSHTTPURLResponse *response, NSError *error) {
switch (response.statusCode) {
case kOCErrorServerPathNotFound:
  //Path not found
  break;
case kOCErrorServerUnauthorized:
  //Bad credentials
  break;
case kOCErrorServerForbidden:
  //Forbidden
  break;
case kOCErrorServerTimeout:
  //Timeout
  break;
default:
  break;
}
}];
```

Move file or folder

Move a file or folder from their current path to a new one on the cloud server. The info needed is the origin path and the destiny path.

Code example

```
[[ AppDelegate sharedOCCommunication ] moveFileOrFolder :sourcePath toDestiny
:destinyPath onCommunication :[ AppDelegate sharedOCCommunication ]

successRequest :^( NSHTTPURLResponse *response, NSString *redirectedServer) {
  //File/Folder moved or renamed
}
failureRequest :^( NSHTTPURLResponse *response, NSError *error) {
  //Failure
  switch (response.statusCode) {
  case kOCErrorServerPathNotFound:
    //Path not found
    break;
  case kOCErrorServerUnauthorized:
    //Bad credentials
    break;
  case kOCErrorServerForbidden:
    //Forbidden
    break;
  case kOCErrorServerTimeout:
    //Timeout
    break;
  default:
    break;
}

}
errorBeforeRequest :^( NSError *error) {
  if (error.code == OCErrorMovingTheDestinyAndOriginAreTheSame) {
    //The destiny and the origin are the same
  }
  else if (error.code == OCErrorMovingFolderInsideHimself) {
    //Moving folder inside himself
  }
  else if (error.code == OCErrorMovingDestinyNameHaveForbiddenCharacters) {
    //Forbidden Characters
  }
  else
  {
    //Default
  }

}];
```

Delete file or folder

Delete a file or folder on the cloud server. The info needed is the path to delete.

Code example

```
[[ AppDelegate sharedOCCommunication ] deleteFileOrFolder :path
onCommunication :[ AppDelegate

sharedOCCommunication ] successRequest :^( NSHTTPURLResponse
__response, NSString__redirectedServer) \{;;
  //File or Folder deleted

} failureRequest :^( NSHTTPURLResponse __response, NSError__error) \{

switch (response.statusCode) \{ case kOCErrorServerPathNotFound:
//Path not found break; case kOCErrorServerUnauthorized: //Bad
credentials break; case kOCErrorServerForbidden: //Forbidden break;
case kOCErrorServerTimeout: //Timeout break; default: break; }

}];
```

Download a file

Download an existing file on the cloud server. The info needed is the server URL, path of the file on the server and localPath, path where the file will be stored on the device and a boolean to indicate if is neccesary to use LIFO queue or FIFO.

Code example

```
NSOperation *op = nil;
op = [[ AppDelegate sharedOCCommunication ] downloadFile :remotePath
toDestiny :localPath withLIFOSystem:isLIFO onCommunication :[ AppDelegate
sharedOCCommunication ]

progressDownload :^( NSUInteger bytesRead, long long totalBytesRead, long long
totalBytesExpectedToRead) {

//Calculate percent
float percent = ( float)totalBytesRead / totalBytesExpectedToRead;
 NSLog ( @"Percent of download: %f" , percent); }
successRequest :^(NSHTTPURLResponse *response, NSString *redirectedServer) {
  //Download complete
}
failureRequest :^(NSHTTPURLResponse *response, NSError *error) {
  switch (response.  statusCode) {
  case kOCErrorServerUnauthorized:
    //Bad credentials
    break;
  case kOCErrorServerForbidden:
    //Forbidden
    break;
  case kOCErrorProxyAuth:
    //Proxy access required
    break;
  case kOCErrorServerPathNotFound:
    //Path not found
    break;
  default:
    //Default
    break;
  }
}
shouldExecuteAsBackgroundTaskWithExpirationHandler :^{
  [op cancel ];
}];
```

Download a file with background session

Download an existing file storaged on the cloud server using background session, only supported by iOS 7 and higher.

The info needed is, the server URL: path where the file is stored on the server; localPath: path where the file will be stored on the device; and NSProgress: object where get the callbacks of the upload progress.

To get the callbacks of the progress is needed use a KVO in the progress object. We add the code in this example of the call to set the KVO and the method where catch the notifications.

Code example

```
NSURLSessionDownloadTask *downloadTask = nil;

NSProgress *progress = nil;

downloadTask = [_sharedOCCommunication downloadFileSession:serverUrl
toDestiny:localPath defaultPriority:YES onCommunication:_sharedOCCommunication
withProgress:&progress successRequest:^(NSURLResponse *response, NSURL
*filePath) {
       //Upload complete
    } failureRequest:^(NSURLResponse *response, NSError *error) {

       switch (error.code) {
          case kCFURLErrorUserCancelledAuthentication:
             //Authentication cancelled
          break;

          default:
             switch (response.statusCode) {
                case kOCErrorServerUnauthorized :
                   //Bad credentials
                break;
                case kOCErrorServerForbidden:
                   //Forbidden
                break;
                case kOCErrorProxyAuth:
                   //Proxy access required
                break;
                case kOCErrorServerPathNotFound:
                   //Path not found
                break;
                default:
                   //Default
                break;
             }
          break;
       }
 }];

// Observe fractionCompleted using KVO
 [progress addObserver:self forKeyPath:@"fractionCompleted"
options:NSKeyValueObservingOptionNew context:NULL];


//Method to catch the progress notifications with callbacks
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context
{
    if ([keyPath isEqualToString:@"fractionCompleted"] && [object
```

```
isKindOfClass:[NSProgress class]]) {
        NSProgress *progress = (NSProgress *)object;

        float percent = roundf (progress.fractionCompleted * 100);

        //We make it on the main thread because we came from a delegate
        dispatch_async(dispatch_get_main_queue(), ^{
            NSLog(@"Progress is %f", percent);
        });
    }
}
```

Set callback when background download task finishes

Method to set callbacks of the pending download transfers when the app starts. It's used when there are pendings download background transfers. The block is executed when a pending background task finishes.

Code example

```
[[AppDelegate sharedOCCommunication] setDownloadTaskComleteBlock:^NSURL
*(NSURLSession *session, NSURLSessionDownloadTask *downloadTask, NSURL
*location) {


}];
```

Set progress callback with pending background download tasks

Method to set progress callbacks of the pending download transfers. It's used when there are pendings background download transfers. The block is executed when a pending task get a input porgress.

Code example

```
[[AppDelegate sharedOCCommunication]
setDownloadTaskDidGetBodyDataBlock:^(NSURLSession *session,
NSURLSessionDownloadTask *downloadTask, int64_t bytesWritten, int64_t
totalBytesWritten, int64_t totalBytesExpectedToWrite) {


}];
```

Upload a file

Upload a new file to the cloud server. The info needed is localPath, path where the file is stored on the device and server URL, path where the file will be stored on the server.

Code example

```

```

```
NSOperation *op = nil;
op = [[ AppDelegate sharedOCCommunication ] uploadFile :localPath toDestiny :
remotePath onCommunication :[ AppDelegate sharedOCCommunication ]

progressUpload :^( NSUInteger bytesWrote, long long totalBytesWrote, long long
totalBytesExpectedToWrite) {
  //Calculate upload percent
  if ( totalBytesExpectedToRead/1024 != 0) {
    if ( bytesWrote > 0) {
     float percent = totalBytesWrote* 100 / totalBytesExpectedToRead;
      NSLog ( @"Percent: %f" , percent);
    }
  }
}
successRequest :^( NSHTTPURLResponse *response, NSString *redirectedServer) {
  //Upload complete
}
failureRequest :^( NSHTTPURLResponse *response, NSString *redirectedServer,
NSError *error) {
  switch (response.  statusCode) {
  case kOCErrorServerUnauthorized :
    //Bad credentials
    break;
  case kOCErrorServerForbidden:
    //Forbidden
    break;
  case kOCErrorProxyAuth:
    //Proxy access required
    break;
  case kOCErrorServerPathNotFound:
    //Path not found
    break;
  default:
    //Default
    break;
  }
}
failureBeforeRequest :^( NSError *error) {
  switch (error.code) {
    case OCErrorFileToUploadDoesNotExist:
      //File does not exist
      break;
    default:
      //Default
      break;
  }
}
shouldExecuteAsBackgroundTaskWithExpirationHandler :^{
  [op cancel];
}];
```

Upload a file with background session

Upload a new file to the cloud server using background session, only supported by iOS 7 and higher.

The info needed is localPath, path where the file is stored on the device and server URL, path where the file will be stored on the server and NSProgress object where get the callbacks of the upload progress.

To get the callbacks of the progress is needed use a KVO in the progress object. We add the code in this example of the call to set the KVO and the method where catch the notifications.

Code example

```
NSURLSessionUploadTask *uploadTask = nil;

NSProgress *progress = nil;

uploadTask = [[AppDelegate sharedOCCommunication] uploadFileSession:localPath
toDestiny:remotePath onCommunication:[ AppDelegate sharedOCCommunication ]
withProgress:&progress successRequest:^(NSURLResponse *response, NSString
*redirectedServer) {
        //Upload complete
    } failureRequest:^(NSURLResponse *response, NSString *redirectedServer,
NSError *error) {
        switch (response.statusCode) {
    case kOCErrorServerUnauthorized :
      //Bad credentials
      break;
    case kOCErrorServerForbidden:
      //Forbidden
      break;
    case kOCErrorProxyAuth:
      //Proxy access required
      break;
    case kOCErrorServerPathNotFound:
      //Path not found
      break;
    default:
      //Default
      break;
    }

  }];

// Observe fractionCompleted using KVO
 [progress addObserver:self forKeyPath:@"fractionCompleted"
options:NSKeyValueObservingOptionNew context:NULL];
```

```
//Method to catch the progress notifications with callbacks
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context
{
    if ([keyPath isEqualToString:@"fractionCompleted"] && [object
isKindOfClass:[NSProgress class]]) {
        NSProgress *progress = (NSProgress *)object;

        float percent = roundf (progress.fractionCompleted * 100);

        //We make it on the main thread because we came from a delegate
        dispatch_async(dispatch_get_main_queue(), ^{
            NSLog(@"Progress is %f", percent);
        });

    }
}
```

Set callback when background task finish

Method to set callbacks of the pending transfers when the app starts. It's used when there are pendings background transfers. The block is executed when a pending background task finished.

Code example

```
[[AppDelegate sharedOCCommunication]
setTaskDidCompleteBlock:^(NSURLSession *session, NSURLSessionTask *task,
NSError *error) {


}];
```

Set progress callback with pending background tasks

Method to set progress callbacks of the pending transfers. It's used when there are pendings background transfers. The block is executed when a pending task get a input porgress.

Code example

```
[[AppDelegate sharedOCCommunication]
setTaskDidSendBodyDataBlock:^(NSURLSession *session, NSURLSessionTask *task,
int64_t bytesSent, int64_t totalBytesSent, int64_t totalBytesExpectedToSend) {



}];
```

Check if the server supports Sharing api

The Sharing API is included in ownCloud 5.0.13 and greater versions. The info needed is activeUser.url, the server URL that you want to check.

Code Example

```
[[ AppDelegate sharedOCCommunication ] hasServerShareSupport :_activeUser.url
onCommunication :[ AppDelegate sharedOCCommunication ]

  successRequest :^( NSHTTPURLResponse *response, BOOL hasSupport, NSString
*redirectedServer) {
  }
  failureRequest :^( NSHTTPURLResponse *response, NSError *error){
  }
}];
```

Read shared all items by link

Get information about what files and folder are shared by link.

The info needed is Path, the server URL that you want to check.

Code example

```
[[ AppDelegate sharedOCCommunication ] readSharedByServer :path
onCommunication :[ AppDelegate sharedOCCommunication ]

successRequest :^( NSHTTPURLResponse *response, NSArray *items, NSString
*redirectedServer) {
  NSLog ( @"Item: %d" , items);
}

failureRequest :^( NSHTTPURLResponse *response, NSError *error){
  NSLog ( @"error: %@" , error);
  NSLog ( @"Operation error: %d" , response.statusCode);
}];
```

Read shared items by link of a path

Get information about what files and folder are shared by link in a specific path.

The info needed is the server URL that you want to check and the specific path tha you want to check.

Code example

```
[[AppDelegate sharedOCCommunication] readSharedByServer:serverPath
andPath:path onCommunication:[AppDelegate sharedOCCommunication]
successRequest:^(NSHTTPURLResponse *response, NSArray *items, NSString
*redirectedServer) {
        NSLog ( @"Item: %d" , items);


    } failureRequest:^(NSHTTPURLResponse *response, NSError *error) {
        NSLog ( @"error: %@" , error);
        NSLog ( @"Operation error: %d" , response.statusCode);
}];
```

Share link of file or folder

Share a file or a folder from your cloud server by link. The info needed is Path, your server URL and the path of the item that you want to share (for example /folder/file.pdf)

Code example

```
[[ AppDelegate sharedOCCommunication ] shareFileOrFolderByServer :path
andFileOrFolderPath :itemPath onCommunication :[ AppDelegate
sharedOCCommunication ]
successRequest :^( NSHTTPURLResponse *response, NSString *token, NSString
*redirectedServer) {

NSString *sharedLink = [ NSString stringWithFormat:@
`path/public.php?service=files&t=%@
<mailto:path/public.php?service=files&t=%25@>`_
, token];

}
failureRequest :^( NSHTTPURLResponse *response, NSError *error){
  [ _delegate endLoading ];

DLog ( @”error.code: %d” , error.  code);
DLog (@”server.error: %d”, response.  statusCode);
int code = response.  statusCode ;
if (error.code == kOCErrorServerPathNotFound) {
}

switch (code) {
case kOCErrorServerPathNotFound:
  //File to share not exists
  break;
case kOCErrorServerUnauthorized:
  //Error login
  break;
case kOCErrorServerForbidden:
  //Permission error
```

```
  break;
case kOCErrorServerTimeout:
  //Not possible to connect to server
  break;
default:
if (error.code == kOCErrorServerPathNotFound) {
  //File to share not exists
} else {
  //Not possible to connect to the server
}
break;

}

}];

}

NSLog ( @"error: %@" , error);
NSLog ( @"Operation error: %d" , response.statusCode);
}];
```

Unshare a folder or file by link

Stop sharing by link a file or a folder from your cloud server.

The info needed is Path, your server URL and the Id of the item that you want to Unshare.

Before unsharing an item, you have to read the shared items on the selected server, using the method “ readSharedByServer ” so that you get the array items with all the shared elements. These are objects OCShareDto, one of their properties is idRemoteShared, parameter needed to unshared an element.

Code example

```
[[ AppDelegate sharedOCCommunication ] unShareFileOrFolderByServer :path
andIdRemoteSharedShared :sharedByLink.  idRemoteShared onCommunication :[
AppDelegate sharedOCCommunication ]

  successRequest :^( NSHTTPURLResponse *response, NSString *redirectedServer)
{
    //File unshared
  }
  failureRequest :^( NSHTTPURLResponse *response, NSError *error){
    //Error
  }
];
```

Check if file of folder is shared

Check if a specific file or folder is shared in your cloud server.

Teh info need is Path, your server URL and the Id of the item that you want.

Before check an item, you have to read the shared items on the selected server, using the method " readSharedByServer " so that you get the array items with all the shared elements. These are objects OCShareDto, one of their properties is idRemoteShared, parameter needed to unshared an element.

Code example

```
[[AppDelegate sharedOCCommunication] isShareFileOrFolderByServer:path
andIdRemoteShared:_shareDto.idRemoteShared onCommunication:[AppDelegate
sharedOCCommunication] successRequest:^(NSHTTPURLResponse *response,
NSString *redirectedServer, BOOL isShared) {
    //File/Folder is shared

  } failureRequest:^(NSHTTPURLResponse *response, NSError *error) {
    //File/Folder is not shared
}];
```

Tips

- Credentials must be set before calling any method
- Paths must not be on URL Encoding
- Correct path: https://example.com/owncloud/remote.php/dav/Pop_Music/
- Wrong path: https://example.com/owncloud/remote.php/dav/Pop%20Music/
- There are some forbidden characters to be used in folder and files names on the server, same on the ownCloud iOS library /,<,>,:,",\`,?,*
- To move a folder the origin path and the destination path must end with /
- To move a file the origin path and the destination path must not end with /
- Upload and download actions may be cancelled thanks to the object NSOperation
- Unit tests, before launching unit tests you have to enter your account information (server url, user and password) on OCCommunicationLibTests.m

Bugtracker

Thank you for helping ownCloud by reporting bugs. Before submitting an issue, please read Issue submission guidelines first.

- If the issue is with the ownCloud server, report it to the Core repository
- If the issue is with the ownCloud desktop client, report it to the Desktop client repository
- If the issue is with the ownCloud iOS app, report it to the iOS repository
- If the issue is with the ownCloud Android app, report it to the Android repository
- If the issue with with an ownCloud app, report it to where that app is developed
- If the issue is with a Marketplace app, report it to the Marketplace issue tracker
- If the app is listed in our main github repository report it to the correct sub repository

- If the app is listed in the apps repository report it there

Please note that the mailing list should not be used for bug reports, as it is hard to track them there.

Code Reviews

"" Given enough eyeballs, all bugs are shallow ""

Introduction

In order to increase the code quality within ownCloud, developers are requested to perform code reviews. As we are now heavily using the GitHub platform these code review shall take place on GitHub as well.

Precondition

From now on no direct commits/pushes to master or any of the stable branches are allowed in general. **Every code** change - **even one liners** - have to be reviewed!

How will it work?

1. A developer will submit his changes on GitHub via a pull request (PR). GitHub:help - using pull requests
2. Within the pull request the developer could already name other developers (using

   @GitHubusername) and ask them for review.
3. Using Labels section on the right side, they add `*3 - To review*` label if the patch is complete. If they have no permission to do that, other developers may add this Label in case PR author had indicated.
4. Other developers (either named or at free will) have a look at the changes and are welcome to write comments within the comment field.
5. In case the reviewer is okay with the changes and thinks all his comments and suggestions have been take into account a :+1 on the comment will signal a positive review.
6. Before a pull request will be merged into master or the corresponding branch at least 2 reviewers need to give :+1 score.
7. Our continuous integration server will give an additional indicator for the quality of the pull request.

Examples

Read our coding style guidelines for information on what a good pull request and good ownCloud code looks like.

These are two examples that are considered to be good examples of how pull requests should be handled

- https://github.com/owncloud/core/pull/121
- https://github.com/owncloud/core/pull/146

Questions?

Feel free to drop a line on the mailing list or join us on IRC.

Bug Triaging

Bug Triaging is the process of checking bug reports to see if they are still valid (the problem might be solved since the bug was reported), reproducing them when possible (to make sure it really is an ownCloud issue and not a configuration problem) and in general making sure the bug is useful for a developer who wants to fix it. If the bug is not useful and can't be augmented by the original reporter or the triaging contributor, it has to be closed.

Why do you want to join

Helping to bring the number of issues down makes it easier for developers to spend their time productively and bug triagers thus **contribute greatly to ownCloud development**! Triaging a bug doesn't take long so the work comes in small chunks and you don't need many skills, just some patience and sometimes perseverance.

Bug triagers who contribute significantly should ask to be listed as an active contributor on the owncloud.org page!

How do you triage bugs

The process of checking, reproducing and closing invalid issues is called 'bug triaging'. Issues can be divided in one of three kinds:

1. Bugs or feature requests which come with all needed information to allow a developer to fix or work on them
2. Incomplete or duplicate bug reports or feature requests
3. Irrelevant or wrong bug reports or feature requests

The job of a bug triager is to identify the One's for developers to look at, help remove, merge or improve any Two to a One and dismiss Three's in a friendly and emphatic way.

Triaging follows these steps:

- Find an issue somebody should look at
- Be that somebody and see if the issue content is useful for a developer
- Reply and close, ask a question, add information or a label.
- Find the next bug-to-be-dealt-with and repeat!

General considerations

- You need a github account to contribute to bug triaging.
- If you are not familiar with the github issue tracker interface (which is used by ownCloud to handle bug reports), you may find this guide useful.
- You will initially only be able to comment on issues. The ability to close issues or assign labels will be given liberally to those who have shown to be willing and able to contribute. Just ask on IRC!
- Read our bug reporting guidelines so you know what a good report should look like and where things belong. The issue template asks specifically for some information developers need to solve issues.
- It might even be fixed, sometimes! It can also be fruitful to contact the developers on irc. Tell them you're triaging bugs and share what problem you bumped into. Or just ask on the test-pilots mailing list.
- To ensure no two people are working on the same issue, we ask you to simply add a comment like I am triaging this in the issue you want to work on, and when done,

before or after executing the triaging actions, note similarly that you're done.

To be able to tag and close issues, you need to have access to the repository. For the core and sync app repositories this also means having signed the contributor agreement. However, this isn't really needed for triaging as you can comment after you're done triaging and somebody else can execute those actions.

Finding bugs to triage

Github offers several search queries which can be useful to find a list of bugs which deserve a closer look:

- The bugs least recently commented on
- Least commented issues
- Bugs which need info

But there are more methods. For example, if you are a user of ownCloud with a specific setup which uses Apache as the webserver, Dropbox as storage, or uses the encryption app, you could look for bugs with these keywords. You can then use your knowledge of your installation and your installation itself to see if bugs are (still) valid or reproduce them.

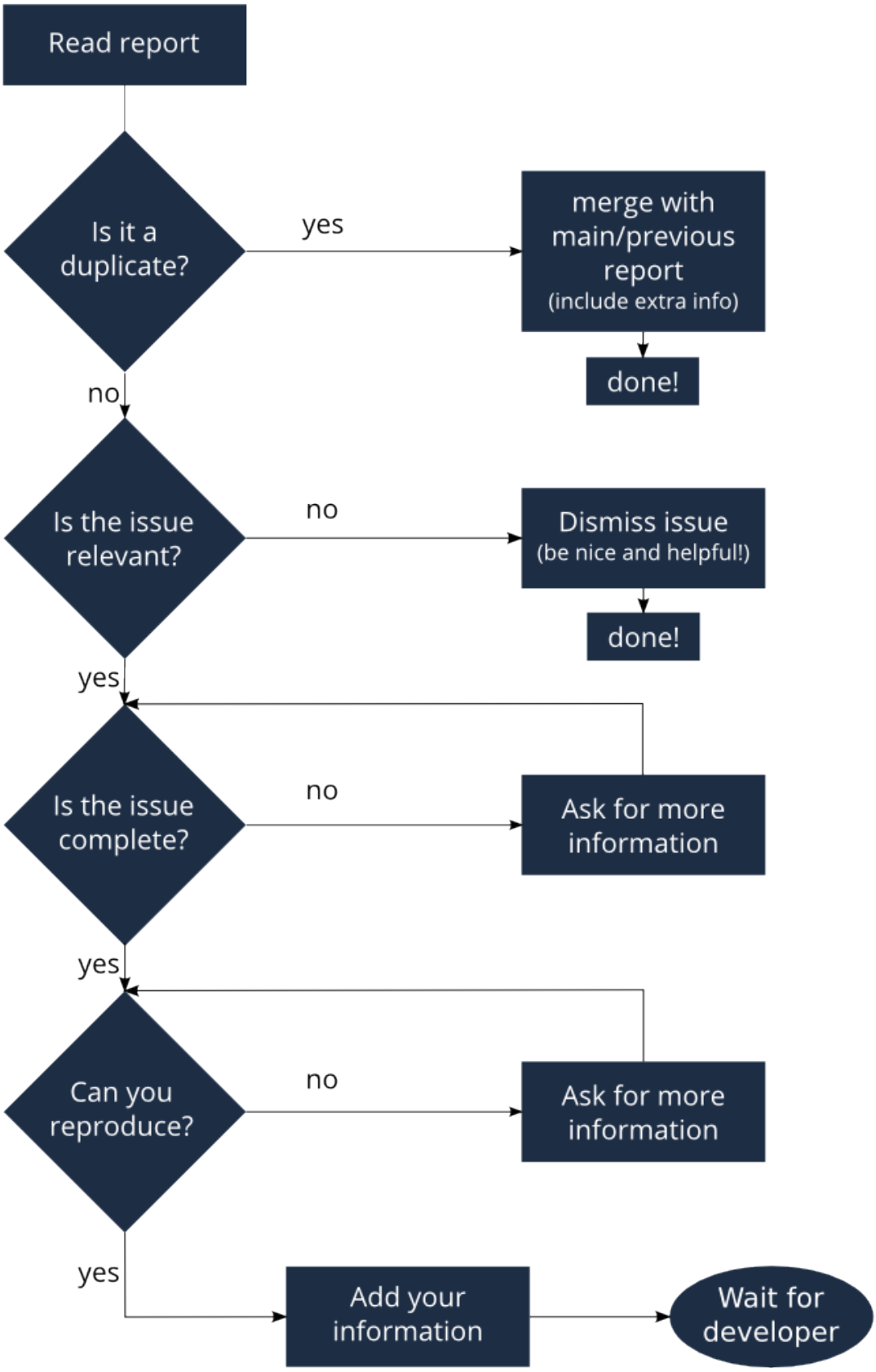
Once you have picked an issue, add a comment that you've started triaging:

I am triaging this bug

Checking if the issue is useful

Much content from Guidelines and HOWTOs/Bug triaging

The goal of triaging is to have only useful bug reports for the developers. And you don't have to know much to be able to judge at least some bug reports to be less than useful. There are duplications, incomplete reports and so on. Here is the work flow for each bug:



Let's go over each step.

Finding duplicates

To find duplicates, the search tool in github is your first stop. In this screen you can easily search for a few keywords from the bug report. If you find other bugs with the same content, decide what the best bug report is (often the oldest or the one where one or more developers have already started to engage and discuss the problem). That

is the `master' bug report, you can now close the other one (or comment that it can be closed as duplicate).

If the bug report you were reviewing contains additional information, you can add that information to the `master' bug report in a comment. Mention this bug report (using #<bug report number>) so a developer can look up the original, closed, report and perhaps ask the initial reporter there for additional information.

If you can't find anything, look in closed bug reports. The problem might be solved already and be listed there! Of course, these other bug reports might be closed as duplicates of the one you are looking at now - if you can't find one that is solved nor can find any duplicates, you can move on to the next step. If you are unsure, just add a comment: might be a duplicate of #<bug nr here> will usually suffice.

When the issue is a feature request, you can be helpful in the same way: merge related requests by adding information of one to the other and closing the first.

Be polite: when you need to request information or feedback be clear and polite, and you will get more information in less time. Think about how you'd like to be treated, were you to report a bug!

You can answer more quickly and friendly using one of these templates.

Often our github issue tracker is a place for discussions about solutions. Be friendly, inclusive and respect other people's position.

Determining relevance of issue

Not all issues are relevant for ownCloud. Bugs can be due to a specific configuration or unsupported platforms. Raspberry Pi's suffer from SQLite time-outs, NGINX has problems which Apache doesn't, and Microsoft Server with IIS is not well supported. While external issues are not always a reason to close a report, be sure that they are clear: does the user use the `standard' platform? Ask for information if this is missing.

Last but not least, the problem might be due to the user doing something that simply does not work. Your general ownCloud knowledge might be helpful here - if this is the case, you can often swiftly close the issue with a comment about what went wrong.

You might have to say no to some requests, for example when a problem has been solved in a new release but won't become available for the release the reporter is using; or when a solution has been chosen which the reporter is unhappy about. Be considerate. People feel surprisingly strong about ownCloud, and you should take care to explain that we don't aim to ignore them; on the contrary. But sometimes, decisions which benefit the majority of users don't help an individual. The extensibility and open availability of the code of ownCloud is here to relieve the pain of such decisions.

Determining if the report is complete

Now that you know that the bug report is unique, and that is not an external issue, you need to check all the needed information is there.

Check our bug reporting guidelines and make sure bug reports comply with it! The information asked in the issue template is needed for developers to solve issues.

Once you added a request for more information, add a #needinfo tag.

If there has been a request for more information on the report, either by you, a developer or somebody else, but the original reporter (or somebody else who might have the answer) has not responded for 1 month or longer, you can close the issue. Be polite and note that whoever can answer the question can re-open the issue!

Reproducing the issue

An important step of bug triaging is trying to reproduce the bugs, this means, using the information the reporters added to the bug report to force (recreate, reproduce, repeat) the bug in the application.

This is needed in order to differentiate random/race condition bugs of reproducible ones (which may be reproduced by developers too; and they can fix them).

To reproduce an issue, please refer to our testing documents.

If you can't reproduce an issue in a newer version of ownCloud, it is most likely fixed and can be closed. Comment that you failed to reproduce the problem, and if the reporter can confirm (or doesn't respond for a long time), you can close the issue. Also, be sure to add what exactly you tested with - the ownCloud Master or a branch (and if so, when), or did you use a release, and if so - what version?

Finalizing and tagging

Once you are done reproducing an issue, it is time to finish up and make clear to the developers what they can do:

- If it is a genuine bug (or you are pretty sure it is) add the `Bug' tag.
- If it is a genuine feature request (or you are pretty sure it is) add the `enhancement' tag.
- If the issue is clearly related to something specific, @mention a maintainer. examples: @schiesbn for encryption, @blizzz for LDAP, @PVince81 for quota stuff… You can find a list of maintainers here.

Now, the developers can pick the issue up. Note that while we wish we would always pick up and solve problems promptly, not all areas of ownCloud get the same amount of attention and contribution, so this can occasionally take a long time.

Collaboration

You can just get started with bug triaging. But if you want, you can register on the testpilot mailing list and perhaps introduce yourself to testpilots@owncloud.org. On this list we announce and discuss testing and bug triaging related subjects.

You can also join the '#owncloud-testing' channel on irc://freenode.net and https://webchat.freenode.net/, to ask questions but keep in mind that people aren't active 24/7, and it can occasionally take a while to get a response. Last, but not least, ownCloud contributor Jan Borchardt has a great guide for developers and triagers about dealing with issues, including some 'stock answers' and thoughts on how to deal with pull requests.

For further questions or help you can also send a mail to:

- X (IRC: Y)

We are looking forward to working with you!

Credit: this document is in debt to the extensive KDE guide to bug triaging.

Have You Found a Mistake In The Documentation?

If you have found a mistake in the documentation, no matter how large or small, please let us know by creating a new issue in the docs repository.