1 API Documentation JSON-Middleware

- 1 API Documentation JSON-Middleware
- 2 Introduction
- 3 JSON-Frontend
  - 3.1 General description
    - 3.1.1 Path pattern
    - 3.1.2 Valid HTTP methods
    - 3.1.3 Authentication
    - 3.1.4 Request
    - 3.1.5 Response
    - 3.1.6 [Body: JSON] or [Body: data]
    - 3.1.7 JSON-Response (RIA)
  - 3.2 Concepts
    - 3.2.1 Resource
    - 3.2.2 File
    - 3.2.3 Directory
    - 3.2.4 Dead Properties
  - 3.3 API description template
  - 3.4 Stateful methods (session based authorization)
    - 3.4.1 List resources of a directory | list resources of a directory extended with thumbnail information - `list` | `extendedList`
    - 3.4.2 Create directory / collection - `mkcol`
    - 3.4.3 Rename resource - `rename`
    - 3.4.4 Edit properties - `proppatch`
    - 3.4.5 Get properties for resource | get properties extended with thumbnails for resource - `propget` | `extendedPropget`
    - 3.4.6 Copy resource | move resource - `copy` | `move`
    - 3.4.7 Delete resource - `delete`
    - 3.4.8 Search resources | search resources extended with thumbnails - `search` | `extendedSearch`
    - 3.4.9 Get user info - `user/userInfo`
    - 3.4.10 Upload fetch a resource by given url - `uploadByUrl`
  - 3.5 Stateless methods (token based authorization)
    - 3.5.1 Download file | open file - `data/download` | `data/open`
    - 3.5.2 Download multiple files as zip archive - `data/archive`
    - 3.5.3 Download (list) pictures in a directory as MediaRSS - `data/mediaRss`
    - 3.5.4 Download session progress info - `data/progressInfo`
    - 3.5.5 Upload multiple files - `data/upload`
    - 3.5.6 Upload (save) file from Pixlr Editor - `data/uploadPixlr`
    - 3.5.7 ~~Upload check / create session before upload started - `assertSession`~~
    - 3.5.8 Upload pre check files before upload started - `uploadPrecheck`
  - 3.6 New methods still to be implemented...
- 4 documentation tagging
- 5 documentation mounts
- 6 documentation quota
- 7 Glossar

2 Introduction

- Introduction JSON-Middleware

3 JSON-Frontend

For historical reasons the component might also be referred to as one of the following:

- onlinespeicher-frontend
- smartdrive-middleware
- json-proxy
- webdav-json-proxy

For reasons of conciseness we will refer to it throughout this document as **JSON-Frontend**.

3.1 General description

The JSON-Frontend acts as a WebDAV-JSON-bridge between SmartDrive-Proxy (Catacomb-WebDAV-Server) and any kind of JSON-enabled client. The first of which and the one provided in all installations as an integrated part of the product is the *Qooxdoo-Client*. The *Qooxdoo-Client* is a JavaScript-based client which runs within any kind of modern web-browser.

3.1.1 Path pattern

All requests are directed towards a web-resource with an URI that is build according to this scheme:

```
https://<hostname>:<port>/op/<username>/<method>/[resource-path][?<parameters>]
```

The <>-enclosed parts are all mandatory and apply to all the interface-methods provided. The part enclosed by [] does not apply to all provided methods and might be optional for some. A full detailed version of the specific path-scheme **will be provided** as part of every method's description.

3.1.1.1 URI-encoding

The [resource-path] describes the file-path to a resource, which may include any valid UTF-8 character. For this reason the caller has to make sure that all the parts within the path are properly URI-encoded before performing a request.

A directory name of '株主優待のご案内' would have to be encoded as follows to create a new directory:

```
/op/john.doe@gmx.com/mkcol/%E6%A0%AA%E4%B8%BB%E5%84%AA%E5%BE%85%E3%81%AE%E3%81%94%E6%A1%88%E5%86%85
```

For the sake of readability we will only use ASCII-characters in the examples below.

3.1.1.2 Forbidden characters

There are some characters that must not be used within a resource-name. These are the following:

```
: / * \ ? " | < >
```

3.1.2 Valid HTTP methods

The JSON-Frontend provides it's services as either one of the following HTTP-methods:

- POST (primary)
- GET

Methods can be **stateless** or **stateful**:

3.1.2.1 Stateless methods (token based authorization)

Stateless methods (see #StatelessMethods) don't require an active tomcat-session. In this cases the user has to authorize himself by providing a *token* which was handed to him within the response of another method. Methods using this kind of authentication include (but might not be limited to):

- `download`
- `upload`
- `zipDownload`
- `mediaRss`
-

3.1.2.2 Stateful methods (session based authorization)

Stateful methods (see #StatefulMethods) require a active tomcat-session. Requests to those paths will be answered with a HTTP **401 Unauthorized** if no session-id or an invalid one is provided.

The URL pattern has always the following format:

```
/op/<username>/<operation-name>[/<resource-path>][?<parameters>]
```

3.1.3 Authentication

Authentication is configurable and can be done via session authentication or X-UI-header-authentication. In addition to the authentication information via cookie or X-UI-authentication-header a **User-Agent** -header has to be provided when accessing stateful API-methods. Any request without a User-Agent-header will be rejected with a HTTP-status-code of 401.

3.1.3.1 Session Authentication

The JSON-Frontend uses a standard Java Tomcat-session for authenticating a client. Therefore the client will have to provide a valid JSESSIONID as a cookie. Which means that the *Cookie*-header within the HTTP-request will have to contain something like this:

```
JSESSIONID=5279CCC0A09E473720BB71248C28C148.a01d39t11
```

In addition to this there also has to be a request header called "Authentication" which must contain the same session-id. This is due to the fact that web-browsers automatically attach the cookie to any request submitted to a domain, which could lead to a security issue when e.g. clicking on a link provided by an external source.

```
Authentication: 5279CCC0A09E473720BB71248C28C148.a01d39t11
```

3.1.3.2 X-UI-Header Authentication

Authentication also can be done by setting the request header `X-UI-AccountId` with value UAS-account-Id of the user to be authenticated.

An interceptor checks for the request header `X-UI-AccountId` and if given creates and sets an according OnlineSpeicherUser to the session.

Methods supported via X-UI-Header authentication are all methods except the FreeMailOnlineSpeicherInterfaceBeschreibungFreigabeMiddleware methods.

Security issues, protection:

The X-UI-Header authentication must only be used internally and never be exposed to external systems / to the web.

It is guaranteed for the live infrastructure that all X-UI-headers are dropped by the firewall system.

In addition the middleware frontend is protected by the `de.web.common.misc.IsInternalFilter` refusing all non internal requests.

Configuration:

For live the X-UI-Header authentication must always be disabled by configuration! For development usage both X-UI and UAS-authentications can be enabled at the same time.

Excerpt from configuration:

```
# Authentication via X-UI-AccountId header:
#
# true <==> internal ip filter enabled
authentication.xuiAuthentication.enabled: true

# true ==> authentication exclusively via xui header
# false ==> both xui and uas authentications enabled, xui authentication has
preference
authentication.xuiAuthentication.enabled.exclusively: false
```

3.1.4 Request

```
GET  /op/<username>/<operation-name> HTTP/1.1
Host: <hostname>
User-Agent: <user-agent string>
Cookie: JSESSIONID=<session-id>
Authentication: <session-id>
```

3.1.7 JSON-Response (RIA)

The JSON-response will **always** follow this structure (**This applies to versions >= 5.0.0 of the JSON-Frontend**):

```
{
  "status": {
    "code" : "<status_code>",
    "message" : "<error_message>"
    "httpStatus" : <http_status_code>,
  },
  "duration" : <duration_in_ms>,
  "response" : { .. }
}
```

The `response` attribute may be left blank depending on the `status.code` returned. As a general guideline you may expect that the `response` **will always** be empty if the result of the execution is considered erroneous. As for successful execution you may expect the presence of a response-body. The exact behavior **will** be described unambiguously for each method later on in this document.

3.2 Concepts

3.2.1 Resource

A **resource** is the basic element describing a file-system entity. A resource can only be one of two distinct types: **a file** or **a directory**.

Every `resource` is guaranteed to have all of the following attributes:

- `name`
- `lastModified`
- `creationDate`
- `downloadtoken`

A `resource` can optionally have deadProperties (see #DeadProperties #DeadProperties).

A `resource` is called `extended resource` if it has the optional thumbNails information.

The resource's JSON format:

```
    {
        "name" : "test.txt",
        "fileSize" : 123,
        "lastModified" : 1154503380000,
        "downloadtoken": "MtnQLk6QursiGTo5VGhg1186581701",

        /* optional, only if exisiting */
        "deadProperties": {
            "dirtype": "0",
            "position: "5",
            "description": "bla"
        },

        /* optional, only if exisiting */
        "thumbNails": {
            "1": {
                "url": "...",
                "mimeType": "..."
```

```
            },
            "2": {
                "url": "...",
                "mimeType": "..."
            }
        }
    }
```

attribute	description	type	example(s)
`name`	Name of a single resource	`String`	`"flowers.jpg", "readme.txt", "my.directory"`
`lastModified`	Timestamp of the last modification	`Number`	`1234567890`
`creationDate`	Timestamp of creation of the resource	`Number`	`1234567890`
`fileSize`	Size of the file in bytes	`Number`	`1024, 923810`

3.2.2 File

If the *resource* is a file it will have also the following attributes:

- `fileSize`

3.2.3 Directory

If the *resource* is a directory it will include:

- `mimeType` = `"application/directory"`
- `uploadtoken`

3.2.4 Dead Properties

Dead properties are properties that can be attached to a resource. Those properties don't belong to the WebDAV standard itself, but may be added in any user's (or client's) convenience. A dead-property consists of the following parts:

- Property-name
- Value

```
"<property_name>" : "<value>"
```

In contrast to the WebDAV-standard the JSON-interface does not support namespaces. Any property-attached to a resource while be stored within the "WEBDE:"-namespace.

Disclaimer

Currently the API **does not support** editing of dead-properties. Also it does only list a certain set of properties when calling ***list*** or ***propget***. Clients using WebDAV directly might be setting properties that are not provided using the JSON-API.

- Attribut `deadProperties` kann leer oder beliebig viele Parameter haben.
- Attribut `thumbNails` ist optional und nur bei den 'extented'-Varianten relevant

3.2.4.1 `dirtype` dead property

A directory can optionally have also have a `dirtype` dead property which is used to give a certain directory a different icon within ***Qooxdoo-Frontend***.

The following predefined `dirtype` values are supported:

value	meaning
0	File, no Directory!
1	Pictures
2	Movies
3	Music
4	Mounted shared directory
5	Documents
6	Trash
7	Attachments
8	Shared directory

3.2.4.2 dead properties example

Note: **>** is used to highlight the relevant element.

```
{
  name : "my.picture.directory",
  lastModified : 1234567890,
  creationDate : 1234567890,
  mimeType : "application/directory",
  uploadtoken : "flXtyJrtLZIfbbI9mxl71271400848",
  downloadtoken : "0vo7SxiuG9xaMC/T9gus1271400848_TKDBlusGC0HntaTvBL2nOw=="

> "deadProperties": {
>     "dirtype": "1",
> }

}
```

A ***resource*** might be returned when calling one of the following methods:

- `list`
- `extendedList`
- `search`
- `extendedSearch`
- `propget`
- `extendedPropget`

3.3 API description template

Every single method will be described in detail in the following section #StatefulMethods and #StatelessMethods. Every description is devided into several parts:

- Description
- Http Request
  - Method
  - URL Pattern
  - Header
  - Parameters
  - Body
- Http Response
  - Status Codes

Code	Description	Meaning
201	Created	The ... was created successfully
...	...	...

  - Header
  - Body
- Example

The **Description** part explains what the intent of the method is and how it works. The **URL-Pattern** describes which ***path*** to use to call the method. In the **Request** part we take a detailed look on what is needed to perform a proper invocation of the method.

3.4 Stateful methods (session based authorization)

These methods require a active tomcat-session. Requests to those paths will be answered with a HTTP **401 Unauthorized** if no session-id or an invalid one is provided.

3.4.1 List resources of a directory | list resources of a directory extended with thumbnail information - `list` | `extendedList`

Description

Lists the content of a directory containing all resources within the directory itself (level-1). The directory itself is not contained within the response.

- der optionale Parameter löscht den WebDAV-internen Cache (Workaround für Upload-Synchronisation)!
- alle dead properties aus dem webde-Namespace werden als Eigenschaften mit den Präfix **webde**_ zurückgegeben und sind optional
- timestamp: unixTime in miliseconds

URL-Pattern

```
/op/<username>/list/<path_of_directory>
/op/<username>/extendedList/<path_of_directory>
```

Request

Type	GET
Parameters	none

Response

```
[~WebdavResource, ~WebdavResource, ~WebdavResource, ... ]
```

Possible response codes

200	OK
400	general error
404	not found

Example

```
GET /op/john.doe@gmx.com/list/my.picture.directory/vacations/hawaii_2009 HTTP/1.1
[
  {
    "name" : "img_1011.jpg",
    "creationDate" : "1185196228000",
    "lastModified" : "1185196228000",
    "fileSize" : 77248,
    "downloadtoken" : "35amf51H+eckDxH5wsMQ1186646251",
    "thumbNails" :
      {
        "1" :
          {
            "mimeType" : "image/jpeg",
            "url" :
"http://thumbs.web.de/getthumb?p=4ZN8$BD75kvcWtc[..]Blqideuey$vZ$q3i5PVMw__&m=o55Wi
2Ri8H34"
          }
      }
  },
  {
    "name" : "img_1012.jpg",
    "creationDate" : "1185196228000",
    "lastModified" : "1185196228000",
    "fileSize" : 68157,
    "downloadtoken" : "45asf51H+eckDxH5wsMQ1186646251",
    "thumbNails" :
      {
        "1" :
          {
            "mimeType" : "image/jpeg",
            "url" :
"http://thumbs.web.de/getthumb?p=4ZN8$BD75kvcWtc[..]Blqideuey$vZ$q3i5PVMw__&m=o55Wi
2Ri8H34"
          }
      }
  },
  {
    "name" : "img_1013.jpg",
    "creationDate" : "1185196228000",
    "lastModified" : "1185196228000",
    "fileSize" : 82011,
    "downloadtoken" : "s7asd5hH+eckDxH5wsMQ1186646251",
    "thumbNails" :
      {
        "1" :
          {
            "mimeType" : "image/jpeg",
            "url" :
"http://thumbs.web.de/getthumb?p=4ZN8$BD75kvcWtc[..]Blqideuey$vZ$q3i5PVMw__&m=o55Wi
2Ri8H34"
          }
      }
```

```
  }
]
```

3.4.2 Create directory / collection - `mkcol`

Creates a directory / collection.

URL-Pattern

```
/op/<username>/mkcol/<path_of_directory>
```

Request

Method	`GET`
Parameters	none

Response

Body	none

Response-codes

Code	Description	Meaning
201	Created	The directory was created successfully
400	General error	Some unknown error has occurred
403	Forbidden	Directory name contains illegal characters / Parent-directory is read-only
405	Method Not Allowed	Resource with specified name exists already
409	Conflict	Specified parent-directory does not exist
507	Insufficient Storage	Resource limit exceeded (1000 files/directorys per directory / 100.000 resources in total) *

**** This numbers may differ on different installations.***

Example

HTTP-Request

```
GET /op/john.doe@gmx.com/mkcol/my.picture.directory/vacations/cuba_2010 HTTP/1.1
```

HTTP-Response

```
200 OK HTTP/1.1
```

3.4.3 Rename resource - `rename`

Renaming of a resource (file or directory) whilst keeping the file in the same parent-directory.

Internally using the Webdav method `MOVE`, see http://www.webdav.org/specs/rfc4918.html#METHOD_MOVE.

- Es wird NIE überschrieben, bei Konflikt muss ggf. nach Nutzerrückfrage ein "/move" ausgelöst werden

URL-Pattern

```
/op/<username>/rename/<resource_path>
```

Request

Method	`POST`
Parameters	none
Body	`JSON`

Body

```
{
  "newName" : "neu.jpg"
}
```

Response-Codes

Code	Description	Meaning
201 (Created)	The source resource was successfully moved, and a new resource was created at the destination.	
204 (No Content)	The source resource was successfully moved to a pre-existing destination resource.	
403 (Forbidden)	The source and destination URIs are the same.	mwf2do name forbidden, invalid characters?...
409 (Conflict)	A resource cannot be created at the destination until one or more intermediate collections have been created.	
412 (Precondition Failed)	A resource with name 'newName' already exists.	
404	srcpath not found	mwf2do not in webdav spec?!...

Example

HTTP-Request

```
POST /op/john.doe@gmx.com/rename/my.test.directory/my.resource.to.be.renamed
```

Request-Body

```
{
  "newName" : "new.resource.name"
}
```

HTTP-Response

```
201 OK HTTP/1.1
```

3.4.4 Edit properties - `proppatch`

Manipulates the properties (meta data) of a resource.

- der Name kann geändern werden (s. rename)
- alle Eigenschaften mit Präfix **WEBDE:** werden als *dead propery* im Webdav Server abgelegt
- Request:

URL-Pattern

```
/op/<username>/proppatch/<resource_path>
```

Request

Method	POST
Parameters	none
Body	JSON

Body

```
{
  "name" : "neu.jpg",
  "WEBDE:dirtype" : "0"
}
```

Response-Codes

Code	Description	Meaning
200	OK	
404	Not Found	not found

Es können beliebig viele Dead Properties übergeben (und gespeichert) werden. Um die Properties wieder zurück zu bekommen müssen diese im JSON-Frontend einkonfiguriert werden (Spring Konfiguration Bean `webdavMethodsFactory`).

Aktuell werden folgende Properties ausgegeben:

- `dirtype`: Darstellungs Flag für Verzeichnisse
- `position`: Position in der Verzeichnisliste (UNDDU)
- `description`: Beschreibung (UNDDU)

 Der Wert einer dead propery kann 512 Byte Text beinhalten

Example

HTTP-Request

```
POST /op/john.doe@gmx.com/proppatch/my.test.directory/my.document.directory
HTTP/1.1
```

Request-Body

```
{
  "WEBDE:dirtype" : "3"
}
```

Response

```
200 OK HTTP/1.1
```

3.4.5 Get properties for resource | get properties extended with thumbnails for resource - `propget` | `extendedPropget`

Reads the properties / properties extended with thumbnail information defined for a resource.

- es werden allg. bzw. selbstgesetzte Properties pro Resource gelesen. Diese Funktion wurde notwendig, da die Ressource "/" nicht per *list* erreichbar ist. Dadurch kann auch ein günstige Abfrage auf ein Verzeichnis gemacht werden.

URL-Pattern

```
/op/<username>/propget/<resource_path>
/op/<username>/extendedPropget/<resource_path>
```

- get Parameter: thumbNailFormatIds=NUMBER (z.B.: '0,1' Default=1)
- Method: GET

- Response: JSON

Response-Codes

Code	Description	Meaning
200	OK	
404	File Not Found	Path not found

Example

HTTP-Request

```
GET /op/john.doe@gmx.com/propget/my.test.directory/my.document.directory HTTP/1.1
```

HTTP-Response

```
200 OK HTTP/1.1
```

Reponse-Body

```
{
   "name" : "test",
   "lastModified" : 1251231413000,
   "creationDate" : 1251231413000,
   "mimeType" : "application/directory",
   "dirtype" : 3,
   "uploadtoken" : "NU6BswC7Vf7kjDqq4LQs1253256338_rM77ADcuT/qxGkCOlc1mJw==",
   "downloadtoken" : "toh7yiI18JknX8qitGSB1253256338_rM77ADcuT/qxGkCOlc1mJw==",
}
```

3.4.6 Copy resource | move resource - `copy` | `move`

- Kopieren von Dateien / Directories von einer Ebene in eine andere
- overwrite=false: das Zielverzeichnis wird um alle Dateien ergänzt, die nicht mit bereits vorhandenen Dateien im Namens-Konflikt stehen. Konflikte werden im Response zurückgegeben.
- overwrite=true: das Zielverzeichnis wird gelöscht und mit dem Quelldateien ersetzt.

URL-Pattern

```
/op/<username>/copy/<target-directory>
/op/<username>/move/<target-directory>
```

Method	`POST`
Body	`JSON`

Request-Body

```
{
  "srcPath" : "/My Pictures",
  "names" : [
    "funny-dog.gif",
    "flowers.jpg",
    "car.jpg",
    "subdirectory.1",
    "subdirectory.2/trouble.txt"
  ],
  "overWrite" : <false|true>, // merge or override?
}
```

- Response: JSON
- Body: JSON - list of conflicts. empty list <==> no conflicts, all files copied / moved succesfully

```
[
  {
    "name" : "flowers.jpg",
    "response" : "409", // HTTP-Code aus dem DAV-Multistatus
    "status" : "Conflict", // Text aus DAV-Multistatus
    "description" : "", // Text aus DAV:responsedescription
    "isCollection" : false
  },
  {
    "name" : "My Pictures",
```

```
    "response" : "409", // HTTP-Code aus dem DAV-Multistatus
    "status" : "Conflict", // Text aus DAV-Multistatus
    "description" : "", // Test aus DAV:responsedescription
    "isCollection" : true
  }
]
```

Response-Codes

Code	Description	Meaning
200	OK	All resources were copied / moved successfully
207	Multi Status	At least one resource could not be copied / moved, see JSON body conflict list
404	Not found	Target directory was not found

Example (copy)

```
POST /op/john.doe@gmx.com/copy/My%20Documents/targetDir HTTP/1.1
{
  "srcPath" : "/",
  "names" : [
    "/My Documents/myfile.txt"
  ],
  "overWrite" : false
}
```

Response Collision

```
[
  {
    "isCollection" : false,
    "description" : "",
    "status" : "Precondition Failed (412)",
    "name" : "/My Documents/myfile.txt",
    "response" : "412"
  }
]
```

Example (copy - overwrite)

```
POST /op/john.doe@gmx.com/copy/My%20Documents/targetDir HTTP/1.1
{
  "srcPath" : "/",
  "names" : [
    "/My Documents/myfile.txt"
  ],
  "overWrite" : true
}
```

Response

```
200 OK HTTP/1.1
```

Example (move)

```
POST /op/john.doe@gmx.com/move/Meine%20Dokumente/targetDir HTTP/1.1
{
  "srcPath" : "/",
  "names" : [
    "/My Documents/myfile.txt"
  ],
  "overWrite" : false
}
```

3.4.7 Delete resource - `delete`

- es werden im *dirname* die Ressourcen im *names*-Feld gelöscht
- aktuell wird als 'dirname="/", Dateinamen mit Komplettpfad, in der Liste genutzt

URL-Pattern

```
/op/<username>/delete/<target-directory>
```

Method	`POST`
Body	`JSON`

Request-Body

```
{
  "names" : [
    "<relative_path_of_resource_1>",
    "<relative_path_of_resource_2>"
  ]
}
```

Response

- JSON Body :== list of errors, empty when all resources could be deleted sucessfully

```
[
 {
  "name" : "blumen.jpg",
  "response" : "404", // HTTP-Code aus dem DAV-Multistatus
  "status" : "File not found", // Text aus DAV-Multistatus
  "description" : "", // Test aus DAV:responsedescription
 }
]
```

Response-Codes

Code	Description	Meaning
200	OK	All resources were deleted successfully
207	Multi Status	At least one resource could not be deleted, see JSON body error list
404	Not found	Target directory was not found

Example 1: absolute paths

```
POST /op/john.doe@gmx.com/delete/ HTTP/1.1
{
  "names" : [
    "/my.test.directory/my.resource.1",
    "/my.test.directory/my.resource.2"
  ]
}
```

Example 2: relative paths

```
POST /op/john.doe@gmx.com/delete/my.test.directory/ HTTP/1.1
{
  "names" : [
    "my.resource.1",
    "my.resource.2"
  ]
}
```

3.4.8 Search resources | search resources extended with thumbnails - `search` | `extendedSearch`

Bei der Suche handelt es sich um eine Sub-String Suche ohne Wildcards.

- Sucht rekursiv ab einem angegebenen Ordner abwärts

URL-Pattern

```
/op/<username>/search/<resource_path>
/op/<username>/extendedSearch/<resource_path>
```

- get Parameter: thumbNailFormatIds=NUMBER (z.B.: '0,1' Default=1)

Request-Body

```
{
  "queryType" : "names", // später auch "content"
  "queryText" : "blumen",
}
```

- Pfade im Ergebnis sind relativ zum Suchordner der Anfrage

Response-Codes

Code	Description
200	OK
400	General error
404	Not Found

Example

3.4.9 Get user info - `user/userInfo`

Returns traffic- and quota-information as well as the localized directory-names for a user.

3.4.9.1 Traffic- and quota-information

The following traffic- and quota-information is returned:

Key	Description
MaxFileNameLength	The maximum number of UTF-8 charachters that are allowed for defining a filename (including it's path).
MaxFileSize	The maximum number of bytes a single file may have.
MaxFilesPerdirectory	The maximum number of resources that are allowed within a directory.
MaxFileCount	The maximum number of resources allowed within a account.
StorageFileCount	The actual number of resources within the account.

"StorageFreemail" : "0", "TrafficOwnerUsed" : "0", "TrafficUpload" : "0", "TrafficUploadQuota" : "4294967295", "TrafficGuestQuota" : "1073741824", "TrafficGuestUsed" : "0", "StorageSmartDrive" : "3274960", "StorageQuota" : "4294967296", "StorageFotoalbum" : "0", "TrafficOwnerQuota" : "4294967296",

3.4.9.2 Localized directory-names

There exist some default directorys that are created for each user when the account is created. The locale initially provided when creating the user will define which language the directorys are created in.

The following directorys are defined:

Key	Description
ROOT	Root directory. Most of the time '/'.

Key	Description
PICTURE	Picture directory. e.g. 'My pictures', 'Meine Bilder'
DOC	Document directory. e.g. 'Meine Dokumente', 'My documents'
MUSIC	Music directory. e.g. 'Meine Musik', 'My music'
VIDEO	Video directory. e.g. 'My videos', 'Meine Videos'
TRASH	Trash directory. e.g. 'Trash', 'Papierkorb'
ATTACHMENT	Attachment directory. e.g. 'New mail attachments', 'Neue Dateianhänge'

URL-Pattern

```
/op/<username>/user/userInfo
```

Method	GET
Body	none

Response-Body

```
{
   "MaxFileNameLength" : "250",
   "StorageFreemail" : "0",
   "TrafficOwnerUsed" : "0",
   "TrafficUpload" : "0",
   "TrafficUploadQuota" : "4294967295",
   "TrafficGuestQuota" : "1073741824",
   "TrafficGuestUsed" : "0",
   "MaxFileSize" : "1073741824",
   "StorageSmartDrive" : "3274960",
   "MaxFileCount" : "5000",
   "StorageFileCount" : "1091",
   "StorageQuota" : "4294967296",
   "StorageFotoalbum" : "0",
   "TrafficOwnerQuota" : "4294967296",
   "MaxFilesPerdirectory" : "1000",
   "ROOT" : "/",
   "PICTURE" : "/Meine Bilder",
   "MOUNT" : "/Ordner anderer Personen",
   "DOC" : "/Meine Dokumente",
   "VIDEO" : "/Meine Videos",
   "MUSIC" : "/Meine Musik",
   "TRASH" : "/Papierkorb",
   "ATTACHMENT" : "/Neue Dateianlagen"
}
```

Response-Codes

Code	Description
200	OK
400	General error

3.4.10 Upload fetch a resource by given url - `uploadByUrl`

A file will be uploaded asyncronly into the given target resource, fetching file data directly from the given url.

- That way the user can skip downloading/uploading a file from internet and upload directly to smartdrive instead.
- the source url have to be accessable without name/pw and session-cookie
- in the given tag the status of the download will be accessable

- after this request the target resource is created as an 0-byte file and LOCKed.
- The upload is internally done via method PUT and will affect the storage and the upload-traffic of the user.

- Request:
  - Method POST
  - URL Pattern: `/op/<username>/uploadByUrl/<target-directory>/<target-resource-name>`
  - JSON Body
  - ```
    {
        "url" : "<url>",                // the url to get the upload file from

        "overwrite": true|false         // true ==> overwrite existing file; false ==> response code 412 conflict

        "tag": "foo"                    // ayrbitrary value for querying progress, client must assure uniqueness
    }
    ```

- Response:
  - Response Codes:

Code	Description	Meaning
`200 (Ok)`	The async job was scheduled successfully	
`409 (Preconditon failed)`		

  - Response JSON Body with error details in case of 409 (Preconditon failed):
  - ```
    {
        "errorCode":    "404"
        "errorMessage:  "Download file given by url was not found"
        "errorPhase":   "d"
    // preconditon|download|upload: one of {p, d, u}
    }
    ```

With the **progressInfo** function (1.) the download and (2.) the upload can be tracked:

- progressInfo returns one of the following: SCHEDULED, RUNNING, FINISHED, ERROR, UNKNOWN
- errorDetails Object included in JSON in case of ERROR (not implemented yet, proposal)
- ```
  {
      "progress": "ERROR"

      "errorDetails": {
          "errorCode":    "507"
          "errorMessage:  "Storage or upload traffic over quota"
          "errorPhase":   "u"
    // preconditon|download|upload: one of {p, d, u}
      }
  ```
  ```

- }

- Precondition ERROR

Code	Description
p 409	Directory /<target-directory>/ does not exist
p 412	overwrite was false and /<target-directory>/<target-resource-name> already exists
p 423	The /<target-directory>/<target-resource-name> resource was locked.

- Download ERROR

Code	Description
d 401	No rights to access download file given by url
d 404	Download file given by url was not found

- Upload ERROR

Code	Description
u 401	No rights to upload file to /<target-directory>/<target-resource-name>
u 507	Storage or upload traffic over quota

3.4.10.1 Little TK:

1. evaluate request url / request parameters / request body ==> username, target-directory, target-resource-name, url, tag
2. HEAD (check download is available and accessible)
3. start thread, write 'started' progress to memcache
   executed async inside thread:
   GET (download file) | PUT (upload file to webdav) ('pipeing' via stream, permanently writing progress 'n bytes of total m bytes' ? to memcache ...)
4. write response (witten immediatley after thread start)
5. end thread, write 'done' progress to memcache

3.5 Stateless methods (token based authorization)

These kind of methods don't require a active tomcat-session. Authentication and authorization is performed by checking the provided token. There are two kinds of tokens:

- Download-tokens
- Upload-tokens

Download-tokens are used to authorize read-only access to a resource. **Upload-tokens** are used to check for write-permission to a specific directory.

All tokens expire after a certain period of time. For **download- and upload-tokens** the expiration-time is set to **9 hours**.

3.5.1 Download file | open file - `data/download` | `data/open`

This request is stateless and can be executed without the need to provide a session id or authorization-header. The authorization is realized by using a so-called download-token. In simple terms a download-token is a secure-hash on the associated path. In case of the download-token there is an additional timestamp included that renders the token useless after a specified time.

- Die Tokens für den Up/Download einer Datei können aus den Rückgabe Struckturen der Methoden list, search und propget ausgelesen werden. Sie dienen als Authorisierungs Merkmal, nur wer den Token kennt kann die Datei runterladen.
- Über den Konfiguration parameter `expireSeconds` des Beans `uploadTokenService`, `downloadTokenService` (Spring Konfiguration) kann die Gültigkeitsdauer der Token eingestellt werden.

Datei in Clientapplikation (Browser/Word) öffnen

- Request:
  - URL: /data/<username>/download/bild.jpg?token=<xyz>
  - optionaler Parameter: redirectUrl=<url>, wird als Redirekt ausgeführt, wenn die Datei nicht ausgeliefert werden kann. Die ZielURL[2] wird dann mit &error_code=xxx&error_msg=bla" aufgerufen-
  - Method: GET
- Response: download Stream
- der Fileserver liefert keinen Mime-Type (";") aus
- directory können nicht geöffnet werden ("405 Method Not Allowed" vom Fileserver)

Beispiel:

```
GET
/data/testuser/download/Meine+Dokumente%2Ftest.txt&token=tD%2BQaj8IkaPjoJohjtRQ1186
584218
```

- Response: Content als Stream

3.5.2 Download multiple files as zip archive - `data/archive`

Diese Anfrage wird ohne Session erledigt, kann also auf eine andere Instanz angefragt werden.
⚠Encoding Problem: Das Encoding der Dateinamen in einem von der Java API gepackten Zip Archiv ist nicht festgelegt. So kann es sein, Umlaute in den Dateien beim Entpacken nicht richtig wieder hergestellt werden können.

Kann das Zip Archiv nicht vollständig erstellt werden, werden dem Archiv auf oberster Ebene Fehler Dateien angehängt:

- `_ZIP_ERROR_.txt`: Das Erstellen des Zip Archivs konnte nicht abgeschlossen werden
- `_EMPTY_directory_.txt`: Es wurde versucht über ein leeres Verzeichnis ein Archiv zu erstellen.
- `_ERROR_.txt`: Falls eine oder mehrere Dateien nicht zum Archiv hinzugefügt werden konnten (z.B. Quota abgelaufen), werden sie in dieser Datei aufgelistet.

Aufruf:

- Es können mehrere Ressourcen zu einem Archiv zusammenpackt werden.
- Fehler werden unter <tag> gespeichert (s. Download Session /data/<username>/progressInfo), ohne tag keine Infos
- Request:
  - URL: /data/<username>/archive/
  - Method: POST

- POST Parameter :
  - `resourceList` (textarea): <downloadtoken> /<username>/<path>\n<downloadtoken> /<username>/<path>\n<downloadtoken> /<username>/<path>\n
  - `archiveName` name.zip
  - `tag`: <TAG ID zur Fortschrittsüberwachung>
  - `directoryPrefix`: <path der vom Anfang abgeschnitten werden soll>
- Response: Ziparchiv als Stream
- der Fileserver liefert
  - für eine einzelne Datei diese mit Mime-Type "application/octet-stream"
  - für mehrere Ressourcen (z.B. directory den rekursiv verpackten Inhalt) in einem ZIP-Archiv mit Mime-Type "application/zip"

Beispiel:

```
POST /data/&lt;username&gt;/archive/

resourceList=9qvINMojbZ28KA1bNsrD1186584431+%2Fsmartdrive-test-
01%2FMeine+Dokumente%2Ftest%0D%0A&archiveName=test.zip&tag=ZIP+-+TAG_0004-thvo-
native-12022-1186584431.55&directoryPrefix=Meine+Dokumente%2F
```

- Response: Zipdatei als Stream

Test Formular:

```
<html>
<body>
<form method="post" action="http://localhost:8080/data/testuser01/archive">
Verzeichnisse:
<p><textarea name="resourceList" cols="100"
rows="10">YfNFaLTh0fnxvQeEcRZO1186669688 /smartdrive-test-01/gucker.jpg
xxgPP8ryOAocIkRzwW091186669688 /smartdrive-test-01/500MB.png
</textarea></p>
<p>Archiv Name: <input name="archiveName" value="testArchive.zip" type="text"></p>
<p>directory Prefix: <input name="directoryPrefix" value="" type="text"
size="30"></p>
<p>progress Info tag: <input name="tag" value="myzipdownload-123" type="text"></p>
<input value="   Test starten   " type="submit">
</form>
</body></html>
```

download: zipTest.html

3.5.3 Download (list) pictures in a directory as MediaRSS - `data/mediaRss`

MediaRSS is a syndication format developed by Yahoo!, which is used to publish media-content like pictures, videos and audio files. It is a superset of RSS and is based on XML. For more information please consult the MediaRSS Specification.

directory containing pictures will be exported as a MediaRSS stream. Since this interface may be requested by an externally hosted Flash-client like CoolIris it has to be stateless. The MediaRSS-interface is secured by the same token-mechanism as a file-download.

```
/data/&lt;username&gt;/mediaRss/<path>?token=<download_token>&[nocache=<timestamp>]
```

Method	`GET`
URL	=/data/testuser01/mediaRss/my.picture.directory?token=0vo7SxiuG9xaMC

pathpart/parameter	**description**	**example**
`path`	Path to the directory to be listed as MediaRSS[2]	`my.picture.directory`

pathpart/parameter	description	example
token	Download-token for the directory to list	=0vo7SxiuG9xaMC
nocache	(optional) to prevent caching of server-response	random-value to generate a unique URL

```
https://sd2.1und1.de/data/64735136/mediaRss/my.picture.directory?token=0vo7SxiuG9xa
MC/T9gus1271400848_TKDBlusGC0HntaTvBL2nOw==
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:media="http://search.yahoo.com/mrss/" version="2.0">
  <channel>
    <title>MediaRSS directory Output (created with Rome)</title>
    <link>http://united-internet.com</link>
    <description>MediaRSS directory Output for cooliris-flash.</description>
    <item>
      <title>444446.jpg</title>
      <media:thumbnail
url="https://sd2thumbs.1und1.de/getThumb?p=RMnSOGnj[..]y$iqdXg__&amp;m=gxDESX$jf0cF
" />
      <media:content
url="https://sd2thumbs.1und1.de/getThumb?p=RMnSOGnjw[..]CylSmA__&amp;m=56rxqTV9eEvj
" />
    </item>
  </channel>
</rss>
```

3.5.4 Download session progress info - `data/progressInfo`

Read the state (progress) of long running Datapump Operations.

Bei lang andauernden Datenpumpen-Aktionen (ZIP-Downloads bzw. uploadByUrl) kann über einen Tag der Progress aus dem Data-Server gelesen werden. Wird so eine langanhaltende Aktion angestoßen, legt die Middleware unter einer angegebenen Kennung (tag) einen Eintrag im Memcache an, der den Verlauf der Aktion beinhaltet. Über diese Methode lässt sich der Status auslesen.

Grund für diesen Mechanismus ursprünglich für die ZIP-Archiv-Aktion: Das Zip Archiv wird während des Zusammenbaus bereits an den Client gestreamt. Tritt während des Zusammenbaus ein Fehler auf kann dieser nicht an den Client kommuniziert werden. Mit der **progressInfo** Methode hat der client die Möglichkeit abzufragen, ob die Erstellung des Archives erfolgreich war.

Die redirectUrl wird für den Fehlerfall benötigt. Für uploadByUrl ist die redirectUrl optional. Falls sie gesetzt ist, muss sie auf eine gültige Adresse zeigen.

- Request:
  - URL: GET `/data/<username>/progressInfo?tag=<tag>&redirectUrl=<redirectUrl>`
  - Parameters:
    - `tag`: mandatory parameter, arbitrary non empty string, client must provide unique tags for a user.
    - `redirectUrl`: optional parameter, must contain valid url address if given, the request is redirected to the given url
- Response:
  - Response Codes:

Code	Description	Meaning
200	Ok	
400	Bad request, e.g. error in parameters	

- JSON Body, errorDetails Object included in case of ERROR

```
        {
            "progress": "ERROR"

            "errorDetails": {
                "errorCode":    "507"
                "errorMessage:  "Storage or upload traffic
over quota"
                "errorPhase":   "u"
// preconditon|download|upload: one of {p, d, u}
            }
        }
```

- if `redirectUrl` given redirect to the url given in parameter redirectUrl with current `status` (progress) of operation(e.g. ZIP-Download) attached: `<redirectUrl>?status=SCHEDULED|RUNNING|FINISHED|ERROR|UNKNOWN`
- progress is one of the following:
  - `SCHEDULED`: only for asynchronous operations, operation ist scheduled but has not startet yet, is waiting for its excecution
  - `RUNNING`: operation is running
  - `FINISHED`: operation has been finished successfully
  - `ERROR`: operation has been finished with error, **error information included in json body**
  - `UNKNOWN`: no progress information available for the given tag; operation has been finished or not...

3.5.5 Upload multiple files - `data/upload`

Diese Anfrage wird ohne Session erledigt, kann also auf eine andere Instanz angefragt werden.

- Uploadtoken ist zur Authentifizierung notwendig: token=<xyz>
- optionaler Parameter: redirectUrl=<url>, wird als Redirekt ausgeführt, wenn die Datei nicht ausgeliefert werden kann. Die ZielURL[2] wird dann mit &error_code=xxx&error_msg=bla" aufgerufen-
- Request:
  - URL: /data/<username>/upload/
  - Method: POST
  - die Dateien werden als multipart/form-data hochgeladen
  - zusätzliche Post Parameter:

```
uploadToken=<uploadtoken von targetdirectory>
targetdirectory=/<username>/<path>/
```

Sonder-Rolle File Formularname: Jedes Formularfeld hat neben dem Attribut '`filename`' ein weiteres Attribut '`name`'. kommt in diesem Attribut ein Slash ('/') vor, wird der wert als Pfadbestadteil interpretiert. Kommt kein Slash vor wird der wert ignoriert. Der Pfadbestandteil wird folgendermasen in den Dateipfad eingebaut.

```
targetdirectory + name + filename
```

Beispiel:

```
POST /data/testuser01/upload/ HTTP/1.1
Content-Length: 755
Content-Type: multipart/form-data; boundary=----------ThIs_Is_tHe_bouNdaRY_$

------------ThIs_Is_tHe_bouNdaRY_$
Content-Disposition: form-data; name="uploadToken"

vMuMbmtQF0wZHUkP9/Xe1186585214
```

```
------------ThIs_Is_tHe_bouNdaRY_$
Content-Disposition: form-data; name="targetdirectory"

/&lt;username&gt;/Meine Dokumente/
------------ThIs_Is_tHe_bouNdaRY_$
Content-Disposition: form-data; name="file1"; filename="1.txt"
Content-Type: text/plain

file - 1
------------ThIs_Is_tHe_bouNdaRY_$
Content-Disposition: form-data; name="file2"; filename="2.txt"
Content-Type: text/plain

file - 2
------------ThIs_Is_tHe_bouNdaRY_$
Content-Disposition: form-data; name="file3"; filename="3.txt"
Content-Type: text/plain

file - 3
------------ThIs_Is_tHe_bouNdaRY_$--
```

Response:

- 200 OK wenn die Dateien erfolgreich hochgeladen wurden

Formular:

```
<html><body>
<form action="http://localhost:8080/data/smartdrive-test-01/upload"
enctype="multipart/form-data" method="POST">
<p>Target directory: <input name="targetdirectory" value="/smartdrive-test-01/Meine
Bilder/" type="text" size="30"></p>
<p>Upload Token: <input name="uploadToken" value="J5ROCTRrgW4kWJAw6S5R1186671459"
type="text" size="30" ></p>
<p>Datei (1)<input type="file" name="file-1"></p>
<p>Datei (2)<input type="file" name="file-2"></p>
<p>Datei (3)<input type="file" name="file-3"></p>
<p><input type="submit" value="Upload"></p>
</form>
</body></html>
```

download: upload.html

3.5.6 Upload (save) file from Pixlr Editor - `data/uploadPixlr`

Zur Einbindung des Pixlr-Editors wird eine "Eine Datei" - Speichern Funtkion implementiert

Aufruf:

- Fehler werden an die Redir-URL weitergeben
- Request:
  - URL: /data/<username>/uploadPixlr
  - URL-Parameter
    - targetdirectory -
    - token - der Uploadtoken
    - redir - URL nach erfolgreicher Speicherung
  - Method: POST
  - POST: die Datei
- Response:
  - HTTP-Code 302
  - Location: die URL aus dem Parameter redir mit folgenden Parametern:
    - new - der neue Dateiname, wenn "SAVE AS" genutzt wurde
    - old - der alte Dateiname

- code=xxx
- msg=bla

Beispiel:

Request:

```
https://sd2.1und1.de/data/64735136/uploadPixlr?targetdirectory=/64735136/&token=%2F
JTaM4Io5hLd9ScI8E6%2F1266934967_IYyFWu1JDvrsRZYs0PKpzQ%3D%3D&redir=https%3A%2F%2Fsd
2.1und1.de%2Fqxclient%2Fhtml%2Fsave_pixlr.html
```

Response - Location:

```
https://sd2.1und1.de/qxclient/html/save_pixlr.html#old=Blaue+Berge.jpg
```

3.5.7 Upload pre check files before upload started - `uploadPrecheck`

Diese Methode wird von MultiUploadApplet aufgerufen.

Mit der Methode uploadPreCheck kann vor einem Upload überprüft werden ob die Dateien hochgeladen werden können (ein Hinderungsgrund kann fehlende Disk Quota sein oder das Zielverzeichnis ist nicht vorhanden). Zurückgegeben wird eine Liste der angefragten dateien, in der für jede Datei der Status einzeln aufgelistet ist.

- Request:
  - URL: `/data/<username>/uploadPreCheck/`
  - Method: POST
  - die Dateien werden als multipart/form-data hochgeladen
  - Post Parameter:
    - `uploadToken`: <uploadtoken von targetdirectory>
    - `targetdirectory`: <path>/
    - `checkReply (textarea csv)`: checkFile <dateiname>;<dateigröße>;0\ncheckFile <dateiname>;<dateigröße>;\ncheckFile <dateiname>;<dateigröße>;
- die "0" am Ende von checkReply wird nicht ausgewertet
- targetdirectory muss mit einem / enden
- Response:

```
VERSION=<protokoll version>
CHECK_RESULT <Fehlernummer>;<Fehlerbeschreibung>


CHECK_FILE_RESULT <dateiname>;<Fehlernummer>;<Fehlerbeschreibung>;
CHECK_FILE_RESULT <dateiname>;<Fehlernummer>;<Fehlerbeschreibung>;
CHECK_FILE_RESULT <dateiname>;<Fehlernummer>;<Fehlerbeschreibung>;
```

- mit VERSION wird mitgeliefert um welche Version des Protokolls es sich handelt (Aktuell 1.1)

Beispiel:

```
POST /data/dummy_username/uploadPreCheck HTTP/1.1

targetdirectory=%2Fsmartdrive-test-
01%2FMeine+Bilder%2F&uploadToken=T4P1ERPRI5EnwPlENtoi1186672494&checkReply=checkFil
e+1.txt%3B340914%3B0%0D%0AcheckFile+2.txt%3B342077%3B0%0D%0AcheckFile+3.txt%3B34804
6%3B0
VERSION=1.1
CHECK_RESULT 0;kein Fehler


CHECK_FILE_RESULT 1.txt;0;kein Fehler;
CHECK_FILE_RESULT 2.txt;0;kein Fehler;
```

```
CHECK_FILE_RESULT 3.txt;0;kein Fehler;
```

Formular:

```
<html>
<body>
<form method="post"
action="http://localhost:8080/data/dummy_username/uploadPreCheck">
<p>Targetdirectory: <input name="targetdirectory" value="/smartdrive-test-01/Meine
Bilder/" type="text" size="30"></p>
<p>Upload Token: <input name="uploadToken" value="T4P1ERPRI5EnwPlENtoi1186672494"
type="text" size="30" ></p>
Verzeichnisse:
<p>
<textarea name="checkReply" cols="100" rows="10">
checkFile 1.txt;340914;0
checkFile 2.txt;342077;0
checkFile 3.txt;348046;0</textarea>
</p>
<input value="   Test starten   " type="submit" width="200">
</form>
</body>
</html>
```

download: uploadPreCheck.html

File upload check status codes

```
STATUSCODE_OK = 0;
STATUSCODE_FILE_ERROR = 1;
STATUSCODE_directory_NOT_EXISTS = 3;
STATUSCODE_TARGET_IS_NO_directory = 1000;
STATUSCODE_REQUEST_EMPTY = 20;
STATUSCODE_FILE_FILE_EXISTS = 1100;
STATUSCODE_FILE_BAD_FILENAME = 1101;
STATUSCODE_FILE_NOT_ENOUGH_QUOTASPACE = 12;
```

Response-Codes