CF13.1: Content Repository Dokumentation

Inhalt

Konfiguration und Setup

Vor dem Start muss das Plugin konfiguriert werden. Dazu muss im Verzeichnis "`plugins/contentRepoPlugin/src/main/resources`" die Datei "`settings.conf.dist`" als "`settings.conf`" kopiert und diese Datei angepasst werden.

Das Content-Repo Plugin verwendet eine eigene MySQL-Datenbank, um Metadaten zu den angelegten Repos, Verzeichnissen etc. zu speichern. Obwohl es prinzipiell möglich ist, das selbe Datenbank-Schema zu verwenden wie der Simplifier selbst, wird stark empfpohlen ein eigenes Datenbankschema für das Plugin anzulegen, um Namenskonflikte zu vermeiden. Dieses kann allerding ohne Probleme auf dem selben DB-Server liegen, sofern dies die Netzwerktopologie zulässt.

settings.conf

```
database_mysql {
  user: "simplifier"
  pass: "simplifier"
  host: "localhost"
  port: 3306
  database: "contentrepo"
}

...
```

Nachdem die Datenbank-Verbindung konfiguriert wurde, muss das Datenbankschema für das Plugin angelegt werden.

Dazu kann einfach im Plugin-Verzeichnis der SBT/Activator Befehl "`run admin schemify`" ausgeführt werden.

Die benötigten Tabellen und Indizes sollten nun in der konfigurierten MySQL Tabelle angelegt worden sein.

Plugin-Ausführung

Das Plugin befindet sich im Verzeichnis `plugins/contentRepoPlugin` und kann dort mit SBT/Activator über den "`run`" Befehl gestartet werden. Wird über STDIN der Befehl "`stop`" eingegeben, wird die Ausführung des Plugins regulär beendet (und vom AppServer abgemeldet).

Um die Log-Ausgabe zu Konfigurieren (oder in eine Datei auszugeben), kann die Logback-Konfigurationsdatei "`plugins/contentRepoPlugin/src/main/resources/logback.xml` angepasst werden.

Berechtigungen

Das Pdf Plugin stellt das Berechtigungsobjekt "`iTZ_Plugin_ContentRepo`" mit den Ausprägungen (`CreateRepository: Boolean,`

`MimeMappings: Boolean, PermissionObjectID: String, PermissionObjectType: String`) bereit.

Für jeden Slot/HttpSlot Zugriff wird eine berechtigte Rolle mit diesem Berechtigungsobjekt verlangt. Auch normale Slot-Zugriffe, die ohne eine Benutzeridentifikation aufgerufen werden, werden automatisch verweigert, da eine Berechtigung nicht vorliegt.

Eine Berechtigung mit `CreateRepository=true` wird benötigt, um neue Repositories anzulegen.

Eine Berechtigung mit `MimeMappings=true` wird benötigt, um die Zuordnung von MimeTypes zu Extensions verwalten oder einsehen zu können.

Die zusätzlichen String-Ausprägungen (`PermissionObjectType, PermissionObjectID`) bilden ein Berechtigungspaar, mit dem Zugriff auf Repositories, Folders und Files gesteuert wird. Damit ein User mehrere Berechtigungspaare erhalten kann, muss dieser User mehrere Rollen haben, die unterschiedliche Ausprägungen des Berechtigungsobjektes `iTZ_Plugin_ContentRepo` haben.

Um irgendeine Operation (auch lesend) auf einem beliebigen Objekt durchführen zu können, muss ein Berechtigungspaar für dieses Objekt und alle seine Elternelemente vorhanden sein.

Beispiel:

Ein Repository mit einem Berechtigungspaar ("App", "123") enthält ein Verzeichnis mit Berechtigungspaar ("Session", "abc"), welches wiederum eine Datei mit dem Berechtigungspaar ("Session", "abc") enthält.

Um nun diese Datei lesen zu können, wird ein Benutzer mit 2 Rollen benötigt, welche je eine Ausprägung des Berechtigungsobjektes `iTZ_Plugin_ContentRepo` haben mit einmal (`PermissionObjectType = "App", PermissionObjectID = "123"`) und (`PermissionObjectType = "Session", PermissionObjectID = "abc"`).

Um außerdem das Berechtigungsobjekt dieser Datei auf ("App", "456") ändern zu können, wird zusätzlich eine Rolle mit Ausprägung (`PermissionObjectType = "App", PermissionObjectID = "456"`) benötigt.

Datenmodell

Das Plugin verwaltet als Datenobjekte Repositories, Folders, Files und MimeMappings.

MimeMappings

Diese Datenobjekte spezifizieren Zuordnungen von Dateiendungen auf MimeTypes. Diese sind global und von den Repositores, Folders und FIles unabhängig.

Wird einer Dateiendung kein MimeType zugeordnet, so wird automatisch "application/octet-stream" als Default verwendet.

Datenbank Tabelle **mime_mapping**:

Feld	Typ	Bedeutung
mime_mapping_id	INT	Primärschlüssel
ext	VARCHAR	Dateiendung (lowercase, normiert)
mime_type	VARCHAR	Zugeordneter MimeType

Repositories

Ein Repository beschreibt einen Einstiegspunkt für die Verzeichnisstruktur. Ein Repository ist einem ContentProvider zugeordnet, welche entscheidet, wo die Daten für die Dateien des Repositories gespeichert werden sollen. Repos enthalten außerdem eine Konfiguration für den zugewiesenen ContentProvider in Form von zugeordneten Properties.

Zunächst ist nur das ContentRepository "FileSystem" verfügbar, welches Dateien in einem lokalen Verzeichnis abspeichert. Diese Provider benötigt die Property "basedir", welche das lokale Verzeichnis angibt, in welchem die Dateien gespeichert werden sollen.

Datenbank Tabelle **contentrepository**:

Feld	Typ	Bedeutung
content_id	INT	Primärschlüssel

name	VARCHAR	Name des Repos
permission_object_type	VARCHAR	ObjektType des Berechtigungspaars
permission_object_id	VARCHAR	ObjectId des Berechtigungspaars
provider	VARCHAR	Content Provider (verfügbar: "FileSystem")

Datenbank Tabelle **contentrepositoryconfiguration**:

Feld	Typ	Bedeutung
content_configuration_id	INT	Primärschlüssel
content_id	INT	Fremdschlüssel auf `contentrepository`.`content_id`
configkey	VARCHAR	Property-Key der Provider-Konfiguration (im Fall von "FileSystem" ist der Schlüssel "basedir" definiert)
value	VARCHAR	Property-Value der Provider-Konfiguration

Folders

Folders entsprechen Verzeichnissen, in denen Files oder weitere Unter-Folder liegen können. Zudem ist jeder Folder eindeutig einem ContentRepository zugeordnet.

Datenbank Tabelle **contenfolder**:

Feld	Typ	Bedeutung
folder_id	INT	Primärschlüssel
folder_name	VARCHAR	Name des Folders
folder_description	VARCHAR	Beschreibung des Folders
permission_object_type	VARCHAR	ObjektType des Berechtigungspaars
permission_object_id	VARCHAR	ObjectId des Berechtigungspaars
content_id	INT	Zugeordnetes Content Repo, Fremdschlüssel auf `contentrepository`.`content_id`
parent_folder	INT (nullable)	Übergeordneter Folder (optional), Fremdschlüssel auf `contentrfolder`.`folder_id`
security_schema_id	VARCHAR	Sicherheits-Schema: "`public`" falls dieser Folder oder einer seiner Unterfolder öffentlich abrufbare Dateien enthalten darf, "`private`" ansonsten
status_schema_id	VARCHAR	Status Schema (noch kein Konzept vorhanden, wird immer mit "Default" befüllt)
status_id	VARCHAR	Status ID (noch kein Konzept vorhanden, wird immer mit "Default" befüllt)
current_status	VARCHAR	Aktueller Status (noch kein Konzept vorhanden, wird immer mit "Default" befüllt)

Files

Ein File-Objekt entspricht einer virtuellen Datei, welche eindeutig einem FOlder zugeordnet ist, in welchem diese liegt.

Die Daten der Datei werden nicht in der Datenbank gespeichert, sondern vom Content Provider verwaltet, welche dem zugrundeliegenden ContentRepository zu geordnet ist.

Datenbank Tabelle **contentfile**:

Feld	Typ	Bedeutung

file_id	INT	Primärschüssel
folder_id	INT	Übergeordneter Folder, Fremdschlüssel auf `contentrfolder`.`folder_id`
file_name	VARCHAR	Dateiname; Dateiendung ist ausschlaggebend für den zugeordneten MimeType
permission_object_type	VARCHAR	ObjektType des Berechtigungspaars
permission_object_id	VARCHAR	ObjectId des Berechtigungspaars
security_schema_id	VARCHAR	Sicherheits-Schema: "`public`" falls diese Datei öffentlich abrufbar sein soll, sofern alle übergeordneten Folders ebenfalls "`public`" sind, "`private`" ansonsten
ext_file_id	VARCHAR (nullable)	External Storage File-ID, die von ContentProvider verwaltet werden kann (z.B: Id einer in die Cloud hochgeladenen Datei). Vom Provider "FileSystem" nicht verwendet.
status_schema_id	VARCHAR	Status Schema (noch kein Konzept vorhanden, wird immer mit "Default" befüllt)
status_id	VARCHAR	Status ID (noch kein Konzept vorhanden, wird immer mit "Default" befüllt)

Öffentliche Files

Über die Eigenschaft "security_schema_id" kann der Sichtbarkeitsstatus von ContentFolders und ContentFiles gesteuert werden.

Damit eine Datei öffentlich wird müssen folgende Bedingungen erfüllt sind:

- Der Wert `security_schema_id` des ContentFiles is "`public`"
- Der Wert `security_schema_id` des übergeordneten ContentFolders is "`public`"
    - Alle weiteren übergeordneten ContentFolders dieses ContentFolders müssen ebenfalls die Eigenschaft `security_schema_id` = "`public`" besitzen

Dann kann ein Content-File über die PluginAsset-Schnittstelle heruntergeladen werden:

```
http://localhost:8080/client/1.0/PLUGINASSET/contentRepoPlugin/download/${id}.${ext}
```

Hierbei ist `${id}` der Primärschlüssel des ContentFiles. Die Extension `${ext}` ist arbiträr, kann allerdings zum Medientyp der Datei passend gewählt werden (z.B. `.../PLUGINASSET/contentRepoPlugin/download/55.pdf` für eine öffentliche PDF Datei mit ID = 55).

Im Gegensatz zu den REST Schnittstellen wir keine Authentifizierung verlangt. Außerdem wird die Datei direkt binär zurückgegeben (nicht Base64 Codiert in einem JSON), mit dem MimeType der zu der Extension des ContentFiles (Filename in der Datenbank!) passt.

REST Schnittstellen

Es sind REST-Schnittstellen ("HttpSlots") für alle Datenobjekte und alle Operationen vorhanden:

Datenobjekte:

- mimeMapping
- contentRepository
- contentFolder
- contentFile

Operationen:

- Add
- Get
- Edit
- Delete
- List

Der Name des Slots setzt sich direkt zusammen aus Datenobjekt und Operation (z.B. `contentFolderGet`), bei Http-Slots ist außerdem noch der Suffix "Http" angehängt (z.B. `contentFileAddHttp`).

Alle Operationen sind sowohl als normale Plugin-Slots und als Http-Slots (über REST) implementiert, wie sich exakt gleich verhalten. Eingabe und Ausgabe aller Operationen ist stets JSON.

Jede Rückgabe einer Slot-Operation enthält immer mindestens die folgenen Werte:

JSON Key		Type	Bedeutung
`success`		Boolean	Erfolg der Operation; `true` = erfolgreich, `false` = fehlerhaft
`message`	`msgId`	String	ID der Message (Die Erfolgsmeldung und die Fehlermeldung jeder Datenobjekt/Operation Kombination erhält je eine unterschiedliche ID)
	`msgType`	"E" oder "S"	"E" für Error messages, "S" für Success messages
	`msgText`	String	Text der Nachricht ggf. mit Fehlerbeschreibung

Da diese Rückgabewerte vom Format her immer gleich sind, werden diese bei der folgenden REST Beschreibung immer weggelassen.

Mime Mappings

Hinzufügen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**mimeMappingAddHttp**

Input:

Key	Typ	Bedeutung
extension	String	Dateieindung (z.B. "txt")
mimetype	String	Mime Type (z.B. "text/plain")

Beispiel:

```
{
    "extension": "jpg",
    "mimetype": "image/jpeg"
}
```

Output:

Nichts

Auflisten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**mimeMappingListHttp**

Input:

Nichts
Output:

Key		Typ	Bedeutung
mappings		Array	Array über alle Mappings:
	extension	String	Dateieindung (z.B. "txt")
	mimeType	String	Mime Type (z.B. "text/plain")

Beispiel:

```
{
    "mappings": [
        {
            "extension": "jpg",
            "mimeType": "image/jpeg"
        },
        {
            "extension": "png",
            "mimeType": "image/png"
        },
        {
            "extension": "pdf",
            "mimeType":
"application/pdf"
        }
    ]
}
```

Abfragen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**mimeMappingGetHttp**

Input:

Key	Typ	Bedeutung
extension	String	Dateieindung (z.B. "txt")

Beispiel:

```
{
    "extension": "jpg"
}
```

Output:

Key	Typ	Bedeutung
extension	String	Dateieindung (z.B. "txt")
mimeType	String	Mime Type (z.B. "text/plain")

Beispiel:

```
{
    "extension": "jpg",
    "mimeType": "image/jpeg"
}
```

Bearbeiten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**mimeMappingEditHttp**

Input:

Key	Typ	Bedeutung
extension	String	Dateieindung (z.B. "txt")

mimetype	String	Mime Type (z.B. "text/plain")

Beispiel:

```
{
    "extension": "jpg",
    "mimetype": "image/jpeg"
}
```

Output:

Nichts

Löschen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**mimeMappingDeleteHttp**

Input:

Key	Typ	Bedeutung
extension	String	Dateieindung (z.B. "txt")

Beispiel:

```
{
    "extension": "jpg"
}
```

Output:

Nichts

Content Repositories

Hinzufügen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentRepositoryAddHttp**

Input:

Key	Typ	Bedeutung
name	String	Name des Repositories
provider	String	Content Provider (bisher nur "FileSystem" definiert)
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
config	Object	JSON Objekt mit Properties für den Content Provider. Für "FileSystem" is nur der Key "basedir" definiert, der das Root-Verzeichnis für das Repo bestimmt

Beispiel:

```
{
    "permissionObjectType" : "App",
    "permissionObjectID": "DummyApp",
    "provider" : "FileSystem",
    "name": "MyTestRepo",
    "config" : {
        "basedir": "/path/to/repo"
    }
}
```

Output:

Key	Typ	Bedeutung
id	Int	ID des erstellten ContentRepositories

Beispiel:

```
{
    "id": 15
}
```

Auflisten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentRepositoryListHttp**

Es werden nur die Repositories aufgelistet, für die der Benutzer Berechtigungen besitzt.
Input:

Nichts
Output:

Key		Typ	Bedeutung
repositories		Array	Array über alle Repositories:
	id	Int	ID des Repositories
	name	String	Name des Repositories
	permissionObjectType	String	ObjectType des Berechtigungspaars
	permissionObjectID	String	ObjectId des Berechtigungspaars
	provider	String	Content provider
	config	Object	JSON Objekt mit Properties für den Content Provider.

Beispiel:

```
{
    "repositories": [
        {
            "id": 3,
            "name": "MyRepo",
            "permissionObjectType":
"App",
            "permissionObjectID":
"DummyApp",
            "provider": "FileSystem",
            "config": {
                "basedir":
"/path/to/repo1"
            }
        },
        {
            "id": 4,
            "name": "MyRepo2",
            "permissionObjectType":
"Session",
            "permissionObjectID":
"abc",
            "provider": "FileSystem",
            "config": {
                "basedir":
"/path/to/repo2"
            }
        }
    ]
}
```

Abfragen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentRepositoryGetHttp**

Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:

```
{
    "id": 3
}
```

Output:

Key	Typ	Bedeutung
id	Int	ID des Repositories
name	String	Name des Repositories
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars

provider	String	Content provider
config	Object	JSON Objekt mit Properties für den Content Provider.

Beispiel:

```
{
    "id": 3,
    "name": "MyRepo",
    "permissionObjectType": "App",
    "permissionObjectID": "DummyApp",
    "provider": "FileSystem",
    "config": {
        "basedir": "/path/to/repo1"
    }
}
```

Bearbeiten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentRepositoryEditHttp**

Wenn ein Repository bereits ContentFolders enthält, kann "provider" und "config" nicht mehr verändert werden.

Die angegebenen Werte für "provider" und "config" müssen dann identisch zu den gespeicherten Werten in der Datenbank sein.
Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel (ID des Repositories)
name	String	Name des Repositories
provider	String	Content Provider (bisher nur "FileSystem" definiert)
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
config	Object	JSON Objekt mit Properties für den Content Provider.

Beispiel:

```
{
    "id" : 15,
    "permissionObjectType" : "App",
    "permissionObjectID": "DummyApp",
    "provider" : "FileSystem",
    "name": "MyTestRepo",
    "config" : {
        "basedir": "/path/to/repo"
    }
}
```

Output:

Nichts

Löschen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentRepositoryDeleteHttp**

Ein Repository kann nur gelöscht werden, wenn es keine ContentFolders enthält.

Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:

```
{
    "id": 15
}
```

Output:

Nichts

Content Folders

Hinzufügen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFolderAddHttp**

Input:

Key	Typ	Bedeutung
contentId	Int	ID des ContentRepositories des Folders
name	String	Name des Folders
description	String	Beschreibung des Folder
securitySchemeID	String	"`public`": Folder kann potetiell öffentliche Dateien enthalten, "`private`": Alle Dateien sind nicht-öffentlich.
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
parentFolderId	Int	Id des übergeordneten Folders für diesen Folder (Optional), falls dieser als Subfolder angelegt werden soll.

Beispiel:

```
{
    "contentId" : 3
    "name" : "TestFolder",
    "description" : "This is a test
folder",
    "securitySchemeID" : "private",
    "permissionObjectType" : "Session",
    "permissionObjectID" : "abc",
    "parentFolderId" : 5
}
```

Output:

Key	Typ	Bedeutung
id	Int	ID des erstellten ContentFolders

Beispiel:

```
{
    "id": 15
}
```

Auflisten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFolderListHttp**

Input:

Key	Typ	Bedeutung
contentId	Int	ID des Content Repositories, in dem aufgelistet wird
parentFolderId	Int (Optional)	Falls angegeben: Id des Parent Folders in dem gesucht werden soll Falls nicht angegeben: Es werden nur Folder ohne Parent Folder aufgelistet

Beispiel:

```
{
    "contentId": 3,
    "parentFolderId": 5
}
```

Output:

Key		Typ	Bedeutung
folders		Array	Array über alle Folders:
	id	Int	ID des Folders
	name	String	Name des Folders
	description	String	Beschreibung des Folders
	permissionObjectType	String	ObjectType des Berechtigungspaars
	permissionObjectID	String	ObjectId des Berechtigungspaars
	securitySchemeID	String	Sicherheits-Schema ("public"/"private")
	statusSchemeID	String	Status Schema (Bisher nicht implementiert; immer "Default")
	currentStatus	String	Status Schema (Bisher nicht implementiert; immer "Default")
	statusID	String	Status Schema (Bisher nicht implementiert; immer "Default")

Beispiel:

```
{
    "folders": [
        {
            "id": 5,
            "name": "TestFolder",
            "description": "This is a
test folder",
            "statusSchemeID":
"Default",
            "statusID": "Default",
            "securitySchemeID":
"public",
            "currentStatus": "Default",
            "permissionObjectType":
"Session",
            "permissionObjectID": "abc"
        },
        {
            "id": 7,
            "name": "TestFolder2",
            "description": "This is a
test folder",
            "statusSchemeID":
"Default",
            "statusID": "Default",
            "securitySchemeID":
"private",
            "currentStatus": "Default",
            "permissionObjectType":
"Session",
            "permissionObjectID": "abc"
        }
    ]
}
```

Abfragen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFolderGetHttp**

Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:

```
{
    "id": 3
}
```

Output:

Key	Typ	Bedeutung
id	Int	ID des Folders
name	String	Name des Folders

description	String	Beschreibung des Folders
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
securitySchemeID	String	Sicherheits-Schema ("public"/"private")
statusSchemeID	String	Status Schema (Bisher nicht implementiert; immer "Default")
currentStatus	String	Status Schema (Bisher nicht implementiert; immer "Default")
statusID	String	Status Schema (Bisher nicht implementiert; immer "Default")
parentFolderId	Int (Optional)	ID des übergeordneten Folders (falls vorhanden)

Beispiel:

```
{
    "id": 5,
    "name": "TestFolder",
    "description": "This is a test
folder",
    "statusSchemeID": "Default",
    "statusID": "Default",
    "securitySchemeID": "public",
    "currentStatus": "Default",
    "permissionObjectType": "Session",
    "permissionObjectID": "abc"
}
```

Bearbeiten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFolderEditHttp**

Input:

Key	Typ	Bedeutung
id	Int	ID des Folders
name	String	Name des Folders
description	String	Beschreibung des Folder
securitySchemeID	String	"`public`": Folder kann potetiell öffentliche Dateien enthalten, "`private`": Alle Dateien sind nicht-öffentlich.
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars

Beispiel:

```
{
    "id" : 9
    "name" : "TestFolder",
    "description" : "This is a test
folder",
    "securitySchemeID" : "private",
    "permissionObjectType" : "Session",
    "permissionObjectID" : "abc"
}
```

Output:

Nichts

Löschen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFolderDeleteHttp**

Ein ContentFolder kann nur gelöscht werden, wenn er kein ContentFiles und keine untergeordneten ContentFolders enthält.
Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:

```
{
    "id": 15
}
```

Output:

Nichts

Content Files

Hinzufügen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFileAddHttp**

Input:

Key	Typ	Bedeutung
folderId	Int	ID des übergeordneten Folder
name	String	Dateiname der Datei (wird auch zum bestimmen des MimeTypes verwendet)
securitySchemeID	String	"`public`": Datei ist öffentlich, "`private`": Datei ist nicht-öffentlich.
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
data	String	Base64-Kodierter Inhalt der Datei

Beispiel:

```
{
    "folderId" : 5,
    "name" : "test.txt",
    "securitySchemeID" : "public",
    "permissionObjectType" : "Session",
    "permissionObjectID" : "abc",
    "data" : "dGVzdA=="
}
```

Output:

Key	Typ	Bedeutung
id	Int	ID des erstellten ContentFiles

Beispiel:

```
{
    "id": 15
}
```

Auflisten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFileListHttp**

Input:

Key	Typ	Bedeutung
folderId	Int	ID des Content Folders, in dem aufgelistet wird

Beispiel:

```
{
    "folderId": 3
}
```

Output:

Key		Typ	Bedeutung
files		Array	Array über alle Files:
	id	Int	ID des Files
	name	String	Dateiname des Files
	permissionObjectType	String	ObjectType des Berechtigungspaars
	permissionObjectID	String	ObjectId des Berechtigungspaars
	securitySchemeID	String	Sicherheits-Schema ("public"/"private")
	statusSchemeID	String	Status Schema (Bisher nicht implementiert; immer "Default")
	statusID	String	Status Schema (Bisher nicht implementiert; immer "Default")

Beispiel:

```
{
    "files": [
        {
            "id": 3,
            "name": "test.txt",
            "statusSchemeID":
"Default",
            "statusID": "Default",
            "securitySchemeID":
"public",
            "permissionObjectType":
"Session",
            "permissionObjectID": "abc"
        },
        {
            "id": 4,
            "name": "test2.txt",
            "statusSchemeID":
"Default",
            "statusID": "Default",
            "securitySchemeID":
"public",
            "permissionObjectType":
"Session",
            "permissionObjectID": "abc"
        }
    ]
}
```


Abfragen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFileGetHttp**

Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:


```
{
    "id": 3
}
```


Output:

Key	Typ	Bedeutung
id	Int	ID des Files
folderId	Int	ID des übergeordneten Folders
name	String	Dateiname des Files
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars

securitySchemeID	String	Sicherheits-Schema ("public"/"private")
statusSchemeID	String	Status Schema (Bisher nicht implementiert; immer "Default")
statusID	String	Status Schema (Bisher nicht implementiert; immer "Default")
data	String	Base64-kodierter Inhalt der Datei

Beispiel:

```
{
    "id": 3,
    "folderId": 5,
    "name": "test.txt",
    "statusSchemeID": "Default",
    "statusID": "Default",
    "securitySchemeID": "public",
    "permissionObjectType": "Session",
    "permissionObjectID": "abc",
    "data": "dGVzdA=="
}
```

Bearbeiten: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFileEditHttp**

Input:

Key	Typ	Bedeutung
id	Int	ID der zu bearbeitenden Dateo
name	String	Dateiname der Datei (wird auch zum bestimmen des MimeTypes verwendet)
securitySchemeID	String	"`public`": Datei ist öffentlich, "`private`": Datei ist nicht-öffentlich.
permissionObjectType	String	ObjectType des Berechtigungspaars
permissionObjectID	String	ObjectId des Berechtigungspaars
data	String	Base64-Kodierter Inhalt der Datei

Beispiel:

```
{
    "id" : 5,
    "name" : "test.txt",
    "securitySchemeID" : "public",
    "permissionObjectType" : "Session",
    "permissionObjectID" : "abc",
    "data" : "dGVzdA=="
}
```

Output:

Nichts

Löschen: http://localhost:8080/client/1.0/PLUGIN/contentRepoPlugin/**contentFileDeleteHttp**

Input:

Key	Typ	Bedeutung
id	Int	Primärschlüssel

Beispiel:

```
{
    "id": 15
}
```

Output:

Nichts