ColorServer 5 REST Interface

Content

0

1 Introduction

ColorServer 5 provides a **Representational State Transfer (**REST**)** API. The API provides software developers and system administrators with control over their ColorServer installation outside of the GUI interface.

The REST API is useful for developers and administrators who aim to integrate the functionality of a ColorServer installation with custom scripts or external applications that access the API via the standard Hypertext Transfer Protocol (HTTP or HTTPS).

API users reference particular resources by means of the URL *paths* (the part of the URL after the host name). The response from the ColorServer API includes links that tie together related resources.

1.1 System Overview

This chapter outlines a short overview of the ColorServer 5 product. For more thorough description please refer to the appropriate User Guide.

The central component of the ColorServer 5 is the **Dispatcher** service. It receives Job Tickets from **Clients**, stores them and updates their status. The Dispatcher further distributes Job Tickets to Workers. Moreover, it maintains a list of submitted Job Tickets and an interface providing information about currently existing Job Tickets to interested observers.

Workers connect to the Dispatcher autonomously and offer their services (e.g. a PDF-to-PDF or Image-to-Image processing). The Dispatcher submits Job Tickets to an appropriate Worker which will do the required processing for a given file. The Worker returns status information about the processing back to the Dispatcher.

Dispatcher may be deployed as a standalone **Server**, and in this case it can be accessed through a **REST interface**. Jobs can be started, supervised and their results examined. Moreover, any color resources configured with the Dispatcher (like Job Templates or Color Profiles) may be listed and examined.

Hotfolder Service is a special type of Client monitoring input folders living in specified **Locations** and starting processing of a specified **Workflow** for each new file appearing in the given input folder.

1.2 Resources Hierarchy

The resources exposed in the REST API are arranged in the following hierarchy

```
+---+--> / (root)
    |
    +--------> summary
    +--------> versions
    +--------> productInfo
    +--------> jobs
    |    |
    |    +-------> clients
    |    +-------> logfile
    |    +-------> inkreport
    |
    +--------> files
    +--------> workers
    +--------> licenses
    +---+----> resources
        |
        +---+--> templates
        |   |
        |   +-------> pdf
        |   +-------> image
        |   +-------> color
        |   +-------> normalization
        |   +-------> types
        |
        +---+--> profiles
        |   |
        |   +-------> icc
        |   +-------> mx
        |
        +---+--> spotcolor
        |   |
        |   +-------> db3
        |   +-------> gradations
        |   +-------> rulesets
        |
        +-------> substitutions
        +-------> locations
        +-------> printers
        |
        +---+--> workflows
        |   |
        |   +-------> pdf
        |   +-------> image
        |   +-------> jt
        |   +-------> states
        |
        +---+--> smartprofiler
        |   |
        |   +-------> documents
        |   |
        |   +---+--> defaults
        |       |
        |       +-------> color
        |       +-------> normalization
        |       +-------> templates
        |       +-------> db3
        |       +-------> workflows
```

The *"jobs"* resource exposes management (starting, pausing, stopping, querying) of ColorServer jobs, *"workers"* exports the list of available workers and thus possible job types. The *"files"* resource is used to upload input files to the server and download the output files of processed jobs. The *"licenses"* resource lists ColorServer licences available in the ColorServer installation

Under the *"resources"* path all the ColorServer specific resource files used in job processing are grouped by their types. See 7 for detailed descriptions of their roles. At the moment this collection contains some resources which should be rather placed at the root. This is likely to be changed in future API versions.

1.3 Introductory Examples

To provide a quick entry into the REST interface we'll begin with an example of a simple interaction with Server comprising of starting a job, polling its status and retrieving the results.

1.3.1 Minimal Example

A frirst we send the following HTTP request to the server:

```
POST
http://colorserver:80/jobs?filePath=\\network_drive\path\file.jpg&templateName=
resample-jpg-template&clientId=myUserId
```

Here we assumed that the input file lies on a globally accessible path, and that we know that an example job template named *"resample-jpg-template"* is installed on the Server. The more general case will be discussed further on.

The server thus responds:

```
201 Created
Location: http://colorserver:80/jobs/{9f1f9366-2e1b-4d73-b92f-73d32de76f1d}
```

Thus the job could be started! We can extract the newly created Job's path from the *Location* header, and subsequently, the status of the running job can be queried with:

```
GET http://colorserver:80/jobs/{9f1f9366-2e1b-4d73-b92f-73d32de76f1d}/status
```

As soon as status indicates that the job processing has finished, e.g.:

```
{
    "progress": 100,
    "status": "finished",
    "status-type": "info"
}
```

The result can be fetched as follows:

```
GET http://colorserver:80/jobs/{9f1f9366-2e1b-4d73-b92f-73d32de76f1d}/result
```

Note that the status type was not *"error"*, thus the processing was successful! Server responds with JSON result data in the following format:

```
200 OK
{
   "outputFiles": [
       {
           "fileId": "{bdbceb4c-6699-4a04-929f-8d372705d73a}",
           "type": "result"
       }
   ],
   "processingLog": {
       "messages": [ ... ]
   },
   "resultCategory": "info",
   "startTime": "23-11-2016 08:24:42.765 GMT",
   "finishedTime": "23-11-2016 08:25:59.278 GMT",
   "informationDescription": "",
}
```

The ID of result file(s) is contained in the *outputFiles* array, and can be used to download the job result from Server:

```
GET http://colorserver:80/files/{bdbceb4c-6699-4a04-929f-8d372705d73a}
Accept: application/octet-stream
```

1.3.2 More General Example

In the previous example we assumed that the input file lies on a globally accessible path, and moreover, that we already know about job templates available on the Server. That will be generally not the case.

First, we need to know what types of jobs can be started on the server. To this purpose we query the templates available on the Server with:

```
GET  http://colorserver:80/resources/templates
```

Server responds with JSON data:

```
200 OK

[
    {
        "description": "Crop Images Template",
        "name": "crop-images-templ-3",
        "href": "resources/templates/image/{2bced842-fcf7-46c4-ba24-b373444e664b}",
        "locked": false,
        "updated": "Fri, 12 Sep 2014 14:55:03 GMT",
        "template-type": "ImageProcessing"
    },
    {
        "description": "Invert Images Template",
        "name": "invert-jpg-template",
        "href": "resources/templates/image/{e84d8c8d-52ba-453d-a82f-50fbe8f062c5}",
        "locked": false,
        "updated": "Fri, 12 Sep 2014 14:55:03 GMT",
        "template-type": "ImageProcessing"
    }
]
```

The returned *"href"* attribute contains the path of the resource. Its last element is the **Resource ID.** It may be optionally used in the subsequent requests to reference the deployed template instead of template's name. Both resource IDs and resource names are guaranteed to be unique.

Next problem to solve is the file access. If the Server lives in another domain we must upload the file with HTTP:

```
PUT http://colorserver:80/files?fileName=somefile.jpg&fileDescr=Jobs_inputFile
Content-Type: application/octet-stream

<... File-body ...>
```

Server responds:

```
201 Created
Location: http://colorserver:80/files/{9f1f9366-2e1b-4d73-b92f-73d32de76f1d}
```

The last part of the returned location is the **Resource ID.** It can be used in the subsequent requests to reference the uploaded file.

Now the more general request for starting a job would be:

```
POST http://colorserver:80/jobs?fileId={9f1f9366-2e1b-4d73-b92f-
73d32de76f1d}&templId={2bced842-fcf7-46c4-ba24-b373444e664b}&clientId=myUserId
```

As can be seen, we used resource IDs instead of both direct file path (which wouldn't be possible here) and the template name.

The job result data in this case will again use the file ID:

```
{
   "outputFiles": [
       {
           "fileId": "{bdbceb4c-6699-4a04-929f-8d372705d73a}",
           "type": "result"
       }
   ...
```

Usig the returned ID the result file can be downloaded from the server using HTTP file transfer:

```
GET http://colorserver:80/files/{bdbceb4c-6699-4a04-929f-8d372705d73a}
Accept: application/octet-stream
```

2 General API Features

The API supports only HTTP 1.1. At the moment no authentication or authorization scheme is supported, as the API is supposed to be used only in trusted environments.

The Server can be configured to use the HTTP protocol over SSL/TSL, i.e. HTTPS. This can be used to completely hide all of the HTTP communication inside of an encrypted channel. Please refer to the appropriate User Guide document for detailed descrition of required configuration steps.

In order to be able to use the ColorServer REST API as described here, the ColorServer installation must contain an appropriate REST API license. Otherwise the server will require an authentication token which is only granted to GMG products.

2.1 Representations

All data is sent and received as JSON. All timestamps are returned in RFC1123 format:

```
 Wed, 09 Feb 1994 22:23:32 GMT
```

2.1.1 Summary Representations

When a list of resources is fetched, the response includes a *subset* of the attributes for that resource. This is the *"summary"* representation of the resource for the API to provide. For performance reasons, the summary representation excludes some computationally expensive attributes. To obtain those attributes, please fetch the *"detailed"* representation.

2.1.2 Detailed Representations

When you fetch an individual resource, the response typically includes *all* attributes for that resource plus links to its subresources. The "detailed" representation of the resource will additionally include the contents of the linked subresources.

2.1.3 Hypermedia

All resources may have one or more *href* properties linking to other resources. These are meant to provide explicit URLs so that API clients don't need to construct URLs on their own. It is highly recommended that API clients use these. Doing so will make future upgrades of the API easier for developers. All URLs are expected to be proper RFC 6570 URI templates.

2.1.4 Resource IDs

Each element of a collection can be identfied by its **Name** or, alternatively, by the **Resource ID**. The ID will be generated by the Server and is guaranteed to be unique. Alternatively, the ID can be suplied by the user of the API. The name has to be provided by the user of the API and has to be unique as well.

Example:

```
 http://localhost:8111/resources/templates/image/{6ee231f0-b406-46ce-85cb-
a5a0da5bfe9a}
 http://localhost:8111/resources/templates/image/Invert%20Images%20Template
```

2.2 Requests

2.2.1 Licensing

In order to be able to use the ColorServer REST API as decribed here, the ColorServer installation must contain an appropriate REST API license.

Moreover, usage of some preinstalled resources is rectricted by licensing options. In this case, the API will allow manipulation of that resource, but Workers will reject usage of this resource for color processing!

For more information please refer to the appropriate User Guide document.

2.2.2 Parameters

Many API methods take optional parameters. For GET request parameters should be passed as an HTTP query string parameter.

For POST, PATCH and PUT requests, parameters should be encoded as JSON with a Content-Type of *application/json*, except when explicitely stated otherwise.

When creating new resources with POST or PUT requests optionally an ID to be used for the resource may be specified. The proposed ID must be unique, i.e. it cannot be used by any resource present on the server!

2.2.2.10 *Cachebusting*

The "_" parameter is always ignored, and thus can be used as a *"cachebuster"*, i.e. to prevent browser and internet cashes from returning old data.

Other unspecified parameters will result in the request being rejected.

2.2.3 Conditional Requests

Most responses return an `ETag` header. Some responses also return a `Last-Modified` header. You can use the values of these headers to make subsequent requests to those resources using the as headers `If-None-Match` (for ETags) and `If-Modified-Since` (for timestamps). If the resource has not changed, the server will return status code *304 Not Modified.* Should both headers be used, `If-None-Match` will be used because ETags are more accurate than RFC 1123 dates used for timestamps (see RFC 7232).

The `ETag` can be used for concurrency control, and namely to check if the client has a current version of a resource via the `If-Match` request header. If a third party changed the resource on the server such that the client's `ETag` doesn't match, the server will return error code *412 Precondition Failed*.

Example:

If the last ETag value returned by server was *"xyzzy"* (including double quotes!), the conditional request header would look as follows:

```
If-None-Match: "xyzzy"
```

2.2.4 Cross Origin Resource Sharing

The API supports Cross Origin Resource Sharing (CORS) for AJAX requests, as specified in the CORS W3C working draft.

ColorServer API provides both simple and preflight support for CORS requests. To enable CORS, you must supply either a wildcard string (i.e. *) or a comma separated list of specific domains in the configuration settings of the server and restart it. For more details please refer to the appropriate User Guide document.

When a GET, POST, PUT, PATCH or DELETE request arrives with the `Origin` header set to a valid domain name, the server will compare the domain against the configured list. If the domain appears in the list or the wild card was set, the server will add the `Access-Control-Allow-Origin` header to the response after processing is complete. If the `Origin` domain did not match a domain in the configured list, the server will respond with *200 OK* and error description data stating that access has been denied. In this case the normal request processing will be skipped.

If the request is a CORS preflight, i.e. the OPTION method is used, `Access-Control-Allow-Method`, `Access-Control-Allow-Headers` and `Access-Control-Allow-Origin` headers will be added after standard OPTION processing. If the `Origin` domain did not match a domain in the configured list, the CORS headers won't be added, and a normal OPTION response will be sent back.

Note: Do nor rely on CORS for security! The sole purpose of CORS support is to make it possible for JavaScript code running in a browser to use the REST-API.

2.2.5 Language Negotiation

Applications can use the `Accept-Language` header to request the client-specific language in Server's responses. In that way application can ensure that the returned error messages will be translated to the desired language. Default language is `en-US`. The codes are conformant with IANA Subtag Language Registry (http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry)

You can get a list of supported languages by using the GET request on API's entry point (see 2.4).

2.2.6 Polling of Collections

As to avoid generation of unnecessary network load, the preferred way of polling the collection resources (e.g. */jobs, /workers, /resources, /resources/templates,* etc.) is through usage of `If-Modified-Since` headers. If there was no change to the collection (e.g. no resources were added, deleted or modified) the Server will return a *304 Not Modified*. Otherwise the regular GET response for that collection will be returned.

As the `If-Modified-Since` header only supports dates with a precision up to seconds (RFC 1123), for very frequently changed collections it may be insufficient. For this reason, the */jobs* and */resources/xxx* collections support polling by using `If-None-Match` header and ETags. This is a basic optimization, for more efficient solution some form of server push is required, which might be added in a future version of the API.

Note that it is possible to poll for changes of all resources, namely by querying the overall resource collection under the */resources* path (8.1.1.1). When this collection reports a change, the specific types of resources can be succesively queried as to detect where exactly the change happened.

2.2.7 Headers

Following headers can be used with every request and will not be separately mentioned in the documentation:

Accept : at the moment only *application/json* is supported with all requests, other cases will be separately documented. Of course */* will also be accepted.
Accept-Language : default language is English (*en-US*)

2.2.8 OPTIONS

Every resource can be queried for opertaion types it supports. The supported operations will be returned in the Allow header.

Example:

```
OPTIONS /jobs HTTP/1.1
Accept: */*
Host: colorserver:8011
```

Response:

```
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Allow: DELETE, GET, PATCH, POST
Date: Thu, 25 Sep 2014 08:25:01 GMT
Content-Length: 0
```

2.3 Error Responses

2.3.1 Status Codes

Following status codes can be returned with every response and will not be separately mentioned in the documentation:

200 OK or 204 No Content – Request completed successfully
404 Not Found – Resource could not be found
400 Bad Request – The request parameters or JSON data are incorrect
401 Unauthorized – The request cannot be served, authentication failed
405 Method Not Allowed – Given HTTP verb cannot be applied to this resource, e.g. attempt to use POST with a GET-only endpoint.
406 Not Acceptable – Requested content type not suported by the Server
500 Internal Server Error – Server encountered a problem. The request is probably valid but needs to be retried later, or a there was an unexpected error in Server's backend.

2.3.2 Client Errors

There are three basic types of client errors on API calls that receive JSON request bodies:

Sending invalid JSON will result in a *400 Bad Request* response.

```
HTTP/1.1 400 Bad Request
Content-Length: 45

{"message":"JSON data have incorrect format"}
```

Sending the wrong type of JSON values will result in a *400 Bad Request* response.

```
HTTP/1.1 400 Bad Request
Content-Length: 140

{"message": "Wrong format of server's JSON data, expected an object."}
```

Sending invalid parameters will result in a *400 Bad Request* response.

```
HTTP/1.1 400 Bad Request
Content-Length: 149

{
  "message": "Unsupported request parameter."
  "name": " "workerType"
}
```

Sending invalid parameter value will result in a *400 Bad Request* response.

```
HTTP/1.1 400 Bad Request
Content-Length: 149

{
  "message": "Unsupported request parameter value."
  "name": " "colorSpace",
  "value": " "xxx"
}
```

Other *Bad Request* error responses from Server will have a similiar structure and contain attributes describing the detailed error cause. They will not be specified in this document.

2.4 REST API protocol version

2.4.1 General

To notice HTTP client implementations of changes of REST paths and/or JSON array and object structures a simple versioning has been introduced with ColorServer 5.0.6.

A client can query the `/versions` path to get a JSON array of protocol version objects.

Example:

```
[
    {
        "number": "1",
        "state": "current",
        "stateDetails": "no limitations"
    }
]
```

The array will contain all supported protocol versions of the REST API.

The *number* parameter is the value of the protocol version.

The *state* parameter defines if a protocol version is the latest or a deprecated one. The latest version has the String value "*current*". A deprecated version has a concatenated String value of "*deprecated*" and the date of expiry separated by a space character.

Example:

```
"state": "deprecated Wed, 01 Jan 2020 00:00:00 GMT"
```

The *stateDetails* parameter gives textual information about the changes between the previous and the actual protocol version.

Future ColorServer Server applications will communicate on the same request-response messages structure for a specific protocol version – modifications such as move/rename/change of paths and parameters in JSON response only will happen in a new version. HTTP requests without a specific version will always respond with the latest version available.

Note: From protocol version 2 on all JOSN response messages contain a *version* id parameter, which describes the version of the JSON data set on current message structure level. This version has a different meaning than the protocol version: it is the value of data model version (JSON).

Note: For JSON parameters that are added to the response message no new protocol version will be introduced. Implementations must properly parse the JSON because parameter serialization order in JSON may change.

2.4.2 How to query a specific protocol version?

Versioning feature is accessible via the HTTP path. Protocol versions are Integer numbers. The version number path component must contain the character "v" as a prefix. The protocol version path component must be the first component in the path.

Example:

```
http://colorserver:8111/v1/resources
```

This will query the resources path for protocol version 1 and respond the JSON data structure for this specific protocol version.

Note: The absence of the protocol version path component will respond with the latest version of the REST API.

The error code for an unavailable version request follows the same principle as for a wrong URL path. The server will respond with a "*400 Bad Request*".

3 Entry Point

The user can begin interacting with the API by querying the entry point URI, consisting of a *host*, *base* and *port*:

```
host:<port>/base/
```

Note: *base* will be used for versioning in the future. Leaving it empty will invoke the current default version of the API. In the following examples we assume that *base* is always empty and, for better readibility, that the host name is *colorserver*.

Thus the API entry point is located at:

```
colorserver:<port>/
```

3.1 Elements

3.1.1 GET

Returns an overview information about the deployed ColorServer instance.

Parameters:

summary: here a single counter from the *summary* subcollection can be queried, allowed parameters values are the corresponding counter names of 3.2.1

style [summary | detailed]: default is *summary*, if *detailed* used the representation include the *summary* subcollection of 3.2.1

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

apiVersion: supported version of the ColorServer Web-API

links: list of URI's for the subresources

productInfo: product version information

productInfo/configuration: list of product setup options, possible values: *"Color Server Full"* or *"Ink Optimizer Standalone", "Open API"* if REST API usage is licenced

summary: link to the subresource, if *detailed* used, see 3.2.1.

languages: languages supported on the server

Status Codes:

default (see 2.3.1)

Example 1:

```
GET / HTTP/1.1
Accept: application/json
Host: colorserver:8011
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "apiVersion": "1.0",
   "links": {
       "jobs": { "href": "/jobs" },
       "clients": { "href": "/jobs/clients"  },
       "logfile": { "href": "/jobs/logfile"  },
       "inkreport": { "href": "/jobs/inkreport" },
       "workers": { "href": "/workers" },
       "files": { "href": "/files" },
       "licenseTypes": { "href": "/licenses/types" },
       "licenses": { "href": "/licenses" },
       "resources": { "href": "/resources" },
       "templates": { "href": "/resources/templates" },
       "templateTypes":{"href": "/resources/templates/types" },
       "profiles": { "href": "/resources/profiles" },
       "spotcolor": { "href": "/resources/spotcolor" },
       "db3": { "href": "/resources/spotcolor/db3" },
       "gradations": { "href": "/resources/spotcolor/gradations" },
       "rulesets": { "href": "/resources/spotcolor/rulesets" },
       "substitutions": { "href": "/resources/substitutions" },
       "locations": { "href": "/resources/locations" },
       "printers": { "href": "/resources/printers" },
       "workflows": { "href": "/resources/workflows" },
       "hotfolderMonitor": { "href": "/resources/workflows/states" },
       "smartprofiler": { "href": "/resources/smartprofiler" },
       "smartProfilerDefaults": { "href": "/resources/smartprofiler/defaults" },
       "smartProfilerDocuments": { "href": "/resources/smartprofiler/documents" },
       "summary": { "href": "/summary" }
   },
   "productInfo": {
       "configuration": "Color Server Full, Open API",
       "name": "ColorServer",
       "vendor": "GMG GmbH & Co. KG",
       "version": "5.0.xxxx"
   },
   "summary": {
       "href": "/summary"
   },
   "languages": ["en_US", "de_DE", "es_ES", "pl_PL"]
}
```

Example 2:

```
GET /?summary=db3Count
```

Response:

```
{
  "db3Count": 22
}
```

3.2 Subcollections

The API entry point contains one subcollection:

3.2.1 Summary Subcollection

The *"summary"* subcollection returns an overview of job and resource counters.

It is located at:

```
colorserver:<port>/summary
```

3.2.1.10 *GET*

Returns a collection of various server counters.

Parameters: none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:
various counters, see example below

Status Codes:
default (see 2.3.1)

Example:

```
{
  "db3Count": 6,
  "filesCount": 103,
  "gradationsCount": 0,
  "iccProfilesCount": 104,
  "jobsCompleted": 99,
  "jobsRunning": 0,
  "locationsCount": 3,
  "printersCount": 1,
  "mxProfileCount": 659,
  "rulesetsCount": 3,
  "smartprofDefaultsCount": 5,
  "smartprofDocumentsCount": 1,
  "substitutionsCount": 7,
  "supportedWorkerTypes": 2,
  "templatesCount": 23,
  "workflowsCount": 4
}
```

4 Jobs

4.1 Collection

This collection provides a list of jobs present on the Server. It is located at

```
colorserver:<port>/jobs
```

4.1.1 GET

Returns a list of jobs, both running and finished ones. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

clientId: client ID used for starting jobs - if present, only jobs for this client will be returned, otherwise reply contains the full list

style [summary | detailed]: default is *summary*, if *detailed* used the representation will be that of 4.3.1, Example 2.

jobStatus - filters out jobs with given state only, values: *all, running, finished, notRunning, waiting*

Request Headers:

default (see 2.2.7)

If-Modified-Since: used to poll for collection's changes (see 2.2.6), takes the *clientId* parameter into account, i.e. the user can query for changes of jobs of a specific client only!

If-None-Match: also used to poll for collection's changes. It does not rely on RFC 1123 date precision as it uses ETags (see 2.2.3). It takes the *clientId* parameter into account as well.

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

ETag: the entity tag for the collection (i.e. hash of statuses of all existing jobs)

Response JSON Object:

array of JSON objects with following attributes:

description: detailed description of the job

id: job's resource ID (unique, generated by the server or supplied by user)

name: short name of the job

progress: progress percentage for the job, 100% if job finished

status: *"none", "waitingForAccept", "rejected", "waiting" , "running", "cancelling" , "cancelled", "finished"*

status-type: *"none", "info", "warning", "error", "critical"*

Status Codes:

default (see 2.3.1)

304 Not Modified – If the collection and any of its elements wasn't modified since the time provided in the *If-Modified-Since* header, or if the current entity tag does not differ from the value of *If-None-Match* header.

Example 1:

```
GET /jobs HTTP/1.1
Accept: application/json
Host: colorserver:8011
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 04 Jun 2014 15:58:18 GMT

[
    {
        "description": "",
        "id": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
        "name": "\\network_drive\my-dir\some_picture.png",
        "progress": 100,
        "status": "finished",
        "status-type": "info"
    }
]
```

Example 2:

```
GET http://localhost:8111/jobs?style=detailed&clientId=me HTTP/1.1
Connection: Keep-Alive
Host: localhost:8111
If-Modified-Since: Wed, 04 Jun 2014 15:59:30 GMT
```

Response:

```
HTTP/1.1 304 Not Modified
Server: Microsoft-HTTPAPI/2.0
```

4.1.2 POST

Starts a new job on the server for a specific input file using given set of job templates (8.3) with or without request-specific customizations.

When starting a new ColorServer job, the main template describes general options for image or PDF processing. Additionally the color management template adds settings for color transformations and the normalization template controls color normalization settings.

For PDF jobs there is a possibility to use separate sets of color and normalization templates for vector and image components of the data.

Additionally, some attributes in the used job templates may have been „overriden“ with custom values for the duration of a single ColorServer job.

Moreover, sending files to printers can be achieved by starting special „printing“ jobs.

4.1.2.10 *Base Case*

As to keep the API consistent with the file and resources creation cases, for the base case color processing jobs the parameters are to be encoded in the URL. In the extended case, it is possible to specify standard and overriden template parameters as JSON data (see 4.1.2.2).

Parameters: are to be included as request parameters in the URL

the ID to be used for newly created job – optional, has to be appended to the resource path (see Example 2)

clientId – the ID of the request issuer
jobPriority – requested priority, allowed values: *1* (low) to *10* (high) (*Number*)

filePath – path of the input file, must be accessible form the server.
If both client and server are running on same Windows machine, the local paths can be encoded using the pipe character instead of the colon, e.g. *C|\filename* instead of *C:\filename*.

fileId – alternatively: resource ID of a file previously uploaded to the server
Note: Not all versions and configurations of ColorServer 5 will support „linked" files (6.1.2) as job input file! Please consult the User Guide document.

templateName – name of a job template deployed on the server
templateId –alternatively: resource ID of a job template deployed on the server

colorTemplateName – name of a color management template deployed on the server
colorTemplateId – alternatively: resource ID of a color management template deployed on the server
normTemplateName – name of a normalization template deployed on the server
normTemplateId – alternatively: resource ID of a normalization template deployed on the server

- alternatively (only for PDF jobs):

imgColorTemplateName – name of a color management template to be used for image components of the PDF document
imgColorTemplateId – alternatively: resource ID of a color management template to be used for image components of the PDF document
imgNormTemplateName – name of a normalization template to be used for image components of the PDF document
imgNormTemplateId – alternatively: resource ID of a normalization template to be used for image components of the PDF document
vectColorTemplateName – name of a color management template to be used for vector components of the PDF document
vectColorTemplateId – alternatively: resource ID of a color management template to be used for vector components of the PDF document
vectNormTemplateName – name of a normalization template to be used for vector components of the PDF document
vectNormTemplateId – alternatively: resource ID of a normalization template to be used for vector components of the PDF document
Note: the template pairs for the image and vector parts of thePDF document are optional, i.e. it is allowed to specify only one of them and omit the second. In the general case described above, the specified color or normalization template will be applied to both image and vector parts of thePDF document.

Request Headers:
default (see 2.2.7)

Response Headers:

Location: path to the newly created job, contains job's resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully
400 Bad Request – Bad request

Example 1:

```
  POST
http://colorserver:80/jobs?filePath=\\network_drive\path\file.jpg&templateName=
  simple-jpg-template&clientId=myUserId
```

Response:

```
  HTTP/1.1  201 Created
  Server: Microsoft-HTTPAPI/2.0
  Location: http://colorserver:80/jobs/{9bb6751f-4f1b-4d73-b92f-76fac73d32de}
```

Example 2:

```
  POST http://colorserver:80/jobs/{9bb6751f-4f1b-4d73-b92f-76fac73d32de}?filePath=
\\network_drive\path\file.jpg&templateName=simple-jpg-template&clientId=myUserId
```

Response:

```
  HTTP/1.1  201 Created
  Server: Microsoft-HTTPAPI/2.0

 Location: http://colorserver:80/jobs/{9bb6751f-4f1b-4d73-b92f-76fac73d32de}
```

Example 3:

```
  POST http://colorserver:80/jobs?fileId={751f9bb6-4f1b-4d73-b92f-73d32de76fac}
  &templateName=image-template-1&colorTemplateName=color-template-1
  &normTemplateName=norm-template-1&clientId=myUserId
```

Response:

```
  HTTP/1.1  404 Bad Request
  Server: Microsoft-HTTPAPI/2.0
  Content-Type: application/json

  { "message" : "The given input file not found in repository." }
```

4.1.2.20 *Template Customization Case*

This interface allows the user to start jobs using predefined job templates, but at the same time selectively changing some of the templates' attributes.

Parameters:

none

Request Headers:

default (see 2.2.7)

Content-Type: *application/json*

Request JSON Object:

JSON data, the different formats will be explained in the following

Response Headers:

Location: path to the newly created job, contains job's resource ID

Response JSON Object:

none

Status Codes:

201 Created – Request completed successfully

400 Bad Request – Bad request

4.1.2.2.10 Overrides for Image Template Parameters

The job parameters are to be provided in a JSON object containg attributes described in the following paragraph. If not specified differently, the default type of the attribute is *String*:

clientId – the ID of the request issuer

filePath – path of the input file, must be accessible form the server.
If both client and server are running on same Windows machine, the local paths can be encoded using the pipe character instead of the colon, e.g. *C|\filename* instead of *C:\filename*.

fileId – alternatively: resource ID of a file previously uploaded to the server

Note: Not all versions and configurations of ColorServer 5 will support „linked“ files (6.1.2) as job input file! Please consult the User Guide document.

imageTemplate – JSON object containing:

name - name of a job template deployed on the server

id - alternatively: resource ID of a job template deployed on the server

colorTemplate – JSON object containing:

name – name of a color management template deployed on the server

id - alternatively: resource ID of a color management template on the server

mxProfileName – name of an MX profile to be used as color profile

mxProfileId – alternatively: resource ID of an MX profile to be used as color profile

mxCalibrationProfileName – name of an MX profile to be used for calibration

mxCalibrationProfileId – alternatively: resource ID of an MX profile to be used for calibration

iccInProfileName – name of an ICC profile to be used as input ICC profile

iccInProfileId – alternatively: resource ID of an ICC profile to be used as input

ICC profile

iccOutProfileName – name of an ICC profile to be used as output ICC profile

iccOutProfileId – alternatively: resource ID of an ICC profile to be used as output ICC profile

spotColorDatabaseName – name of a DB3 database to be used for spotcolor conversion

spotColorDatabaseId – alternatively: resource ID of a DB3 database to be used for spotcolor conversion

spotColorDatasetName – name of a dataset from the DB3 database to be used for spotcolor conversion, if *spotColorDatabaseName/Id* has been specified, this attribute must be specified too. The color dataset must be contained in the referenced DB3 database!
Note: for convenience, if the color set name contains the "®" character, this character can be omitted, e.g. *"PANTONE PLUS coated"*can be used for *"PANTONE® PLUS coated"*!

spotColorGradationName – name of a gradation file resource to be used in spotcolor processing

spotColorGradationId – alternatively: resource ID of a of a gradation file resource to be used in spotcolor processing

Note: all the *colorTemplate* overrides above assume that there is only one resource of given type in the target color template! For the general case see 4.1.2.2.2.

normalizationTemplate – JSON object containing:

name – name of a normalization template deployed on the server

id - alternatively: resource ID of a normalization template on the server

targetColorSpaceProfileName – name of an ICC profile to be used for target color space

targetColorSpaceProfileId – alternatively: resource ID of an ICC profile to be used for target color space

rgbNormalizationRules – JSON object containing:

inputColorSpaceProfileName – name of an ICC profile to be used for input color space

inputColorSpaceProfileId – alternatively: resource ID of an ICC profile to be used for input color space

renderingIntent – rendering intent to be used, allowed values:
"relative", "perceptual", "saturation", "absolute", "blackPointCompensation",

cmykNormalizationRules – JSON object, structure as in *rgbNormalizationRules*

grayNormalizationRules – JSON object, structure as in *rgbNormalizationRules*

labNormalizationRules – JSON object, structure as in *rgbNormalizationRules*

Example:

```
POST /jobs HTTP/1.1
Conent-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "clientId": "myClientId",

  "imageTemplate" : {
    "id" : "{c2dda196-400f-4320-9dca-a6ed49ccde4f}"
  },

  "normalizationTemplate" : {
    "name" : "normalization-template-instance-1"
    "targetColorSpaceProfileName" : "icc-profile-instance-1",

    "rgbNormalizationRules" : {
      "inputColorSpaceProfileName" : "icc-profile-instance-2",
      "renderingIntent" : "relative"
    },

    "cmykNormalizationRules" : {
      "inputColorSpaceProfileId" : "{77ca8f9a-23b3-4d2a-910d-c9b64b3b748d}",
      "renderingIntent" : "relative"
    },
  },

  "colorTemplate" : {
    "name" : "color-template-instance-1"
    "mxProfileName" : "mx-profile-instance-1",
    "mxCalibrationProfileName" : "mx-profile-instance-2",
    "iccInProfileId" : "{68643ed7-547f-4210-9fad-27dba83cb5bc}",
    "spotColorGradationName" : "gradation-instance-1",
    "spotColorDatabaseName" : "DB3-instance-1",
    "spotColorDatasetName" : "PANTONE PLUS coated"
  }
}
```

4.1.2.2.20 Indexed Overrides for Image Template Parameters

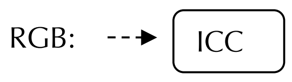
The color template is the most complicated of the templates in ColorServer. It consists of a number of processing steps, in which MX profiles, ICC profiles, a DB3 spotcolor database and spotcolor gradations can be applied to the input with the resulting output forwarded to the consecutive processing step.

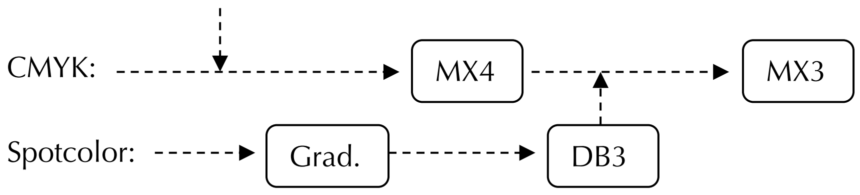
There can be several MX and ICC profiles present in a color template, both applied to the same input type step after step and to different input types as well, because a color template can be applied to several image formats. Because of that, the need for addressing a specific profile for overriding arises. We therefore introduce indexing of processing steps.

The processing step index which is used for overrides is based on the „comes before“ ordering: e.g. if an ICC profile will be applied before an MX profile, it will have a lower index. In order to avoid several corner cases, the indexing is defined by ordering of processing steps in vertical columns of the Color Template Editor in the ColorServer GUI. Indexing starts with 0.

Example:

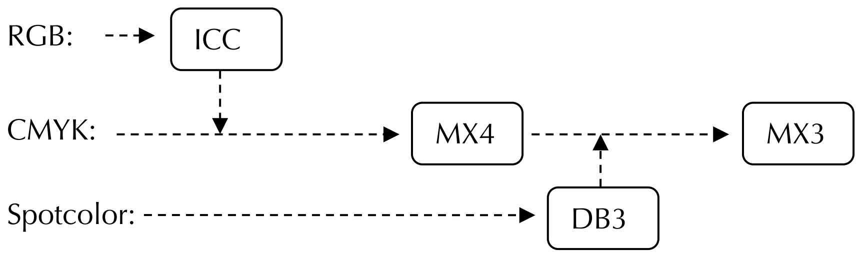
Asume we have following profile configuration:





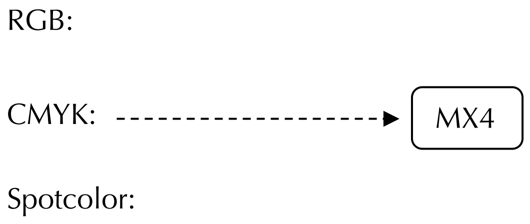
Then by the „comes-before" relation the following ordering of processing steps emerges: ICC:0, Grad:1, MX4:2, DB3:3, MX3:4

In this configuration:



The ordering will be: ICC:0, MX4:1, DB3:2, MX3:3.

And in this one:



The ordering will be of course: MX4:0.

If only a single instance of an ICC or MX processing step is present, usage of indexes is not required. In that case, the data fromat defined in 4.1.2.2.1 will be sufficient, and the single processing step will be found without indexing, even if there are multiple instances of different profile type used in parallel.

Gradation and DB3 processing steps cannot be indexed, as the can be specified only once.

The attributes of the JSON data object are in that case the same as in 4.1.2.2.1, except for *colorTemplate*. If not specified differently, the default type of the attributes is *String*:

- *colorTemplate* – JSON object containing:
  - *name* – name of a color management template deployed on the server
  - *id* - alternatively: resource ID of a color management template on the server

  - *mxProfiles* – array of JSON objects containing:
    - *name* – name of an MX profile to be used as color profile
    - *id*– alternatively: resource ID of an MX profile to be used as color profile
    - *index* – the index of the MX profile processing step to be overridden

mxCalibrationProfiles – array of JSON objects containing:
- *name* – name of an MX profile to be used for calibration
- *id*– alternatively: resource ID of an MX profile to be used for calibration
- *index* – the index of the MX profile processing step to be overridden

iccProfiles – array of JSON objects containing:
- *inName* – name of an ICC profile to be used as input ICC profile
- *inId* – alternatively: resource ID of an ICC profile to be used as input ICC profile
- *outName* – name of an ICC profile to be used as output ICC profile
- *outId* – alternatively: resource ID of an ICC profile to be used as output ICC profile
- *index* – the index of the ICC profile processing step to be overridden

spotColorDatasets – array of JSON objects containing:
- *db3Name* – name of a DB3 database to be used for spotcolor conversion
- *db3Id* – alternatively: resource ID of a DB3 database to be used for spotcolor conversion
- *datasetName* – name of a color dataset from the DB3 database to be used for spotcolor conversion, if *db3Name/db3Id* has been specified, this attribute must be specified too. The color dataset must be contained in the referenced DB3 database!
  Note: for convenience, if the color set name contains the "®“ character, this character can be omitted, e.g. *"PANTONE PLUS coated"* can be used for *"PANTONE® PLUS coated"*!
- *index* – the index of the spotcolor dataset to be overridden.
  Note: in this case it is not an index for color template’s processing steps but a separate index addressing spotcolor datasets used during the DB3 conversion step!

spotColorGradationName – name of a gradation file resource to be used in spotcolor processing, cannot be indexed
spotColorGradationId – alternatively: resource ID of a of a gradation file resource to be used in spotcolor processing, cannot be indexed

Example:

```
POST /jobs HTTP/1.1
Conent-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "clientId": "myClientId",

  "imageTemplate" : {
    "id" : "{c2dda196-400f-4320-9dca-a6ed49ccde4f}"
  },

  "normalizationTemplate" : {
    "name" : "normalization-template-instance-1",
    "targetColorSpaceProfileName" : "icc-profile-instance-1"
  },

  "colorTemplate" : {
    "name" : "color-template-instance-2"

    "mxProfiles" : [
        { "name" : "mx-profile-instance-1", "index" : 1 },
        { "name" : "mx-profile-instance-2", "index" : 3 }
     ],
    "mxCalibrationProfiles" : [
        { "name" : "mx-profile-instance-3", "index" : 4 }
     ],
    "iccProfiles" : [
        { "inId" : "{68643ed7-547f-4210-9fad-27dba83cb5bc}", "index" : 2 }
     ],
    "spotColorDatasets" : [
        { "db3Name" : "DB3-instance-1",
          "datasetName" : "PANTONE PLUS coated",
          "index" : 0 }
     ]
  }
}
```

4.1.2.2.30 Overrides for Standard PDF Template Parameters

The job parameters are to be provided in a JSON object containg attributes described in the following paragraph. If not specified differently, the default type of the attribute is *String*:

clientId – the ID of the request issuer

filePath – path of the input file, must be accessible form the server.
If both client and server are running on same Windows machine, the local paths can be encoded using the pipe character instead of the colon, e.g. *C|\filename* instead of *C:\filename*.

fileId – alternatively: resource ID of a file previously uploaded to the server

Note: Not all versions and configurations of ColorServer 5 will support „linked“ files (6.1.2) as job input file! Please consult the User Guide document.

pdfTemplate – JSON object containing:
name - name of a job template deployed on the server
id - alternatively: resource ID of a job template deployed on the server

pdfX – JSON object containing:
pdfXVersion: PDF/X standard tob e used, allowed values*: "pdfxNone", " pdfx1a2001", "pdfx1a2003", "pdfx32002", "pdfx32002", "pdfx42010"*
isEnabled (Boolean): switches PDF/X output intent on/off
outputIntentIccProfileName: name of an ICC profile to be used for the output intent
outputIntentIccProfileId: alternatively: resource ID of an ICC profile to be used for the output intent
outputIntentConditionText: text to be used for output intent condition
outputIntentInformationText: text to be used for output intent information

colorTemplate – JSON object, same structure as *colorTemplate* in 4.1.2.2.1
normalizationTemplate – JSON object, same structure as *normalizationTemplate* in 4.1.2.2.1

Example:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "filePath": "\\network_drive\path\file.jpg",
  "clientId": "myClientId",

  "pdfTemplate" : {
    "name" : "pdf-base-template-1"
  },

  "pdfX" : {
    "pdfVersion" : " PDF/X-1a:2003"
    "outputIntentIccProfileId " : "icc-profile-instance-1",
    " outputIntentConditionText" : "text text text"
  }
}
```

4.1.2.2.40 Overrides for Extended PDF Template Parameters

The job parameters are to be provided as JSON object containg attributes described in the following paragraph. If not specified differently, the default type of the attribute is *String*:

clientId – the ID of the request issuer
filePath – path of the input file, must be accessible form the server.
If both client and server are running on same Windows machine, the local paths can be encoded using the pipe character instead of the colon, e.g. *C|\filename* instead of *C:\filename*.

fileId – alternatively: resource ID of a file previously uploaded to the server
Note: Not all versions and configurations of ColorServer 5 will support „linked" files (6.1.2) as job input file! Please consult the User Guide document.

pdfTemplate – JSON object, same structure as *pdfTemplate* in 4.1.2.2.3

imageColorTemplate – JSON object, same structure as in *colorTemplate* 4.1.2.2.1, will be used for image data in the PDF document
imageNormalizationTemplate – JSON object, same structure as *normalizationTemplate* in 4.1.2.2.1, will be used for image data in the PDF document
vectorColorTemplate – JSON object, same structure as in *colorTemplate* 4.1.2.2.1, will be used for vector data in the PDF document
vectorNormalizationTemplate – JSON object, same structure as *normalizationTemplate* in 4.1.2.2.1, will be used for vector data in the PDF document

Example:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "clientId": "myClientId",

  "pdfTemplate" : {
    "id" : "{c2dda196-400f-4320-9dca-a6ed49ccde4f}"
  },

  "imageNormalizationTemplate" : {
    "name" : "normalization-template-instance-1"
    "targetColorSpaceProfileName" : "icc-profile-instance-1",
  },

  "vectorNormalizationTemplate" : {
    "name" : "normalization-template-instance-2"
    "targetColorSpaceProfileName" : "icc-profile-instance-2",
  }
}
```

4.1.2.2.50 Compatibility Format

For completness' sake, there's an inteface allowing to start jobs without parameter ovverriding and using same parameter names as in base case (4.1.2.1), but sending them to Server as JSON data. If not specified differently, the default type of the attributes is *String*:

clientId – the ID of the request issuer
jobPriority – requested priority, allowed values: *1* (low) to *10* (high) (*Number*)
filePath – path of the input file, must be accessible form the server.
If both client and server are running on same Windows machine, the local paths can be encoded using the pipe character instead of the colon, e.g. C|*filename* instead of *C:\filename*.

fileId – alternatively: resource ID of a file previously uploaded to the server
Note: Not all versions and configurations of ColorServer 5 will support „linked" files (6.1.2) as job input file! Please consult the User Guide document.

templateName – name of a job template deployed on the server
templateId – alternatively: resource ID of a job template deployed on the server

colorTemplateName – name of a color management template deployed on the server
colorTemplateId – alternatively: resource ID of a color management template deployed on the server
normTemplateName – name of a normalization template deployed on the server
normTemplateId – alternatively: resource ID of a normalization template deployed on the server

- alternatively (only for PDF jobs):

imgColorTemplateName – name of a color management template to be used for image components of the PDF document
imgeColorTemplateId – alternatively: resource ID of a color management template to be used for image components of the PDF document
imgNormTemplateName – name of a normalization template to be used for image components of the PDF document
imgNormTemplateId – alternatively: resource ID of a normalization template to be used for image components of the PDF document
vectColorTemplateName – name of a color management template to be used

for vector components of the PDF document

vectColorTemplateId – alternatively: resource ID of a color management template to be used for vector components of the PDF document

vectNormTemplateName – name of a normalization template to be used for vector components of the PDF document

vectNormTemplateId – alternatively: resource ID of a normalization template to be used for vector components of the PDF document

Example 1:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "clientId": "myClientId",

  "templateName" : "image-template-instance-1",
  "colorTemplateId" : "{c2dda196-400f-4320-9dca-a6ed49ccde4f}",
  "normTemplateId" : "{e2c6bc5c-9503-4a1f-957f-7c572b334713}"
}
```

Example 2:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "clientId": "myClientId",

  "templateName" : "pdf-template-instance-1",
  "imgColorTemplateId" : "{c2dda196-400f-4320-9dca-a6ed49ccde4f}",
  "imgNormTemplateId" : "{e2c6bc5c-9503-4a1f-957f-7c572b334713}"
  "vectColorTemplateName" : "color-template-instance-3",
  "vectNormTemplateName" : " normalization-template-instance-5"
}
```

4.1.2.30 *Printing Case*

It is possible to print a file using the REST API. This can be achieved in two ways: either by directly sending file to a printer, or alternatively by copying the file to a specific „watched" directory, which will than forward the file to the desired printer.

4.1.2.3.10 Direct Printing

Direct printing can be achieved by sending the file to a conceptual printer represented by a printer settings resource as stored in the */resources/printers* collection (see 8.8). If not specified differently, the default type of the attributes is *String*:

Parameters: are to be included as request parameters in the URL

fileId – resource ID of a file previously uploaded to the server

Note: print jobs do not support direct paths as inputs!

clientId – the ID of the request issuer

printerId – resource ID of a printer configuartion

Request Headers:

default (see 2.2.7)

Response Headers:

Location: path to the newly created job, contains job's resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully
400 Bad Request – Bad request

Alternatively, a printing job can be started by sending JSON data tot he server using the same parameter names as in base case for attibute names.

Parameters:

none

Request Headers:

default (see 2.2.7)
Content-Type: *application/json*

Request JSON Object:

fileId – resource ID of a file previously uploaded to the server
Note: print jobs do not support direct paths as inputs!
clientId – the ID of the request issuer
printerId – resource ID of a printer configuartion

Response Headers:

Location: path to the newly created job, contains job's resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully
400 Bad Request – Bad request

Example:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "printerId" : "{7d6757c0-eb88-4f09-82a5-a0a400abba61}",
  "clientId": "myClientId",
}
```

4.1.2.3.20 File Copying

Printing by file copy can be achieved by sending the file to a conceptual directory represented by a shared location resource as stored in the */resources/locations* collection (see 8.7).

Note that choice of the machine, where the target directory is located, will be accomplished by choosing the shared location instance, as share locations are always bound to a specific computer!

If not specified differently, the default type of the attributes is *String*:

Parameters: are to be included as request parameters in the URL
fileId – resource ID of a file previously uploaded to the server
Note: file copy jobs do not support direct paths as inputs!
clientId – the ID of the request issuer
printLocationId – resource ID of a shared location for the file to be copied to
printLocationPath – subpath in the directory indicated by shared location

Request Headers:
default (see 2.2.7)

Response Headers:
Location: path to the newly created job, contains job's resource ID

Response JSON Object:
in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:
201 Created – Request completed successfully
400 Bad Request – Bad request

Alternatively, a file copy job can be started by sending JSON data tot he server using the same parameter names as in base case for attibute names.

Parameters:
none

Request Headers:
default (see 2.2.7)
Content-Type: *application/json*

Request JSON Object:
fileId – resource ID of a file previously uploaded to the server
Note: file copy jobs do not support direct paths as inputs!
clientId – the ID of the request issuer
printLocationId – resource ID of a shared location for the file to be copied to
printLocationPath – subpath in the directory indicated by shared location

Response Headers:
Location: path to the newly created job, contains job's resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully
400 Bad Request – Bad request

Example:

```
POST /jobs HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
  "fileId": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
  "printLocationId" : "{9d947961-de1b-4773-bcee-58254d65024f}",
  "printLocationPath" : "Printer X Input Folder",
  "clientId": "myClientId",
}
```

4.2 Subcollections

The jobs resource path contains two subresources and one subcollection:

4.2.1 Clients Subcollection

The *"clients"* subcollection lets the user query the users of the REST-API which had started jobs on the Server, even if their jobs were deleted at a later time. The list will be resetted when server is restarted.

It is located at:

```
colorserver:<port>/jobs/clients
```

4.2.1.10 *GET*

Returns a list of clients which started jobs on the Server. If not specified differently, the default type of JSON attributes is *String*.

Parameters: none

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

array of client IDs, as supplied by users when starting jobs (see 4.1.2)

Status Codes:

default (see 2.3.1)

Example:

```
[
    "client-1",
    "client-2",
    "client-3"
]
```

4.2.1.20 *Elements*

This collection doesn't have addressable elements - only the list of clients can be requested.

4.2.2 Log File Subresource

The "*logfile*" subresource lets the user download Server's CSV formatted list of the 1000 most recent jobs which were running on the Server.

The maximum size of the log file can be set in Server Configurator, and the file will be then accordingly rewritten in a circular manner.

The resource is located at:

```
colorserver:<port>/jobs/logfile
```

4.2.2.10 *GET*

Returns the file contents oft he CSV formatted Server job log.

Parameters:

none

Request Headers:

default (see 2.2.7)

Accept: *application/octet-stream*

Note: at the moment only raw file representation! The default representation for "*/*" will be *application/octet-stream* as well.

Response Headers:

Content-Type: *application/octet-stream*

Response Body:

Binary file contents.

Status Codes:

default (see 2.3.1)

4.2.3 Ink Report Subresource

The "*inkreport*" subresource lets the user download the ink saving report for the PDF jobs which were running on Server. The calculation of ink savings must however be enabled in a PDF template, and will be carried out <u>only for jobs which are using such templates</u>!

The maximum size of the report file can be set in Server Configurator, and the report file will be then accordingly rewritten in a circular manner.

Additionally, for each job using a template with activated ink saving option <u>a separate ink saving report</u> file will be generated and added to the job's result file set. This file's ID can be then extracted from job's *"result"* subresource (see 4.4.3.1) and then the job specific report can be downloaded from the *"files"* repository.

The resource is located at:

```
colorserver:<port>/jobs/inkreport
```

4.2.3.10 *GET*

Returns the file contents oft he XML formatted ink savings report file.

Parameters:
none

Request Headers:
default (see 2.2.7)
Accept: *application/octet-stream*
Note: at the moment only raw file representation! The default representation for "*/*" will be *application/octet-stream* as well.

Response Headers:
Content-Type: *application/octet-stream*

Response Body:
Binary file contents.

Status Codes:
default (see 2.3.1)

4.3 Elements

Single job instances present at the Server are addressed by their IDs, e.g.:

```
colorserver:<port>/jobs/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
```

4.3.1 GET

Returns the detailed representation of a given Job. If not specified differently, the default type of JSON attributes is *String*.

Parameters:
style [summary | detailed]: default is *summary*, if *detailed* used the contents of subresources will be included

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:
description: detailed description of the job
id: job's unique resource ID generated by the server or supplied by user
name: name of the job generated by the Server
status: link to the subresource, if *detailed* used, see 4.4.1.1 (attributes are

included directly)
result: link to the subresource, if *detailed* used, see 4.4.3.1
ticket: link to the subresource, if *detailed* used, see 4.4.2.1

Status Codes:

default (see 2.3.1)

Example 1 (standard representation):

```
{
    "description": "",
    "id": "{9e3b5c2a-4640-4a31-b12d-369dd2c32522}",
    "name": "some_picture.png",
    "result": {
        "href": "/result"
    },
    "status": {
        "href": "/status"
    },
    "ticket": {
        "href": "/ticket"
    }
}
```

Example 2 (*"detailed"* option used):

```
GET /jobs/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}?style=detailed HTTP/1.1
Accept: application/json
Host: colorserver:8011
```

Response:

```
{
    "description": "",
    "id": "{01fd08bf-ca77-4ca5-9652-1b9ad80f508a}",
    "name": "40004J_023_A000Dc1.pdf",
    "result": {
        "startTime": "23-11-2016 08:24:42.765 GMT"
        "finishedTime": "23-11-2016 08:25:59.278 GMT",
        "resultCategory": "info",
        "informationDescription": "",
        "outputFiles": [
            {
                "fileId": "{4d0f6d63-d026-4393-befb-5f3ccb970455}",
                "type": "result"
            }
        ],
        "processingLog": {
            "messages": [ ... ]
        },
    },
    "status": "finished",
    "statusType": "info",
    "progress": 0,
    "ticket": {
        "clientFilePath": "40004J_023_A000Dc1.pdf",
        "clientId": "Marek.Krajewski@int.gmgcolor.com\\NBDETUSD28",
        "creationTime": "23-11-2016 08:24:42.087 GMT",
        "inputFileId": "{164b7e85-753f-43f2-b9d5-52a5c68ba316}",
        "jobCreator": "Marek.Krajewski",
        "jobParams": {
            "parameters": [ ... ]
        },
        "jobTemplateIds": [
            "{43c3a103-2efd-46ce-90a7-0ef842e4a9b2}",
        ],
        "jobTemplateNames": [
            "No Flattening",
        ],
        "jobTemplateTypes": [
            "PdfProcessing",
        ],
        "jobTicketSupportedOperations": "deleteOnFinished",
        "localeId": "de_DE",
        "subTemplatesTag": [
            {
                "templateId": "PdfProcessing",
                "templateTypeId": "PdfProcessing"
            }
        ],
        "workerTypeId": "PdfProcessing"
    }
}
```

4.3.2 DELETE

Deletes a job on the server. It should be always possible to delete a job.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
none

Response JSON Object:
none

Status Codes:
default (see 2.3.1)

4.4 Subresources

4.4.1 Status

Allows dedicated polling of a job status only. It is located at

```
colorserver:<port>/jobs/:job-id/status
```

4.4.1.10 *GET*

Returns job's current status. If not specified differently, the default type of JSON attributes is *String*.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:

progress: progress percentage for the job, 100% if job finished (Number)

status: *"none", "waitingForAccept", "rejected", "waiting" , "running", "cancelling" , "cancelled", "finished"*

statusType: *"none", "info", "warning", "error", "critical"*

Status Codes:

default (see 2.3.1)

Example:

```
{
    "progress": 100,
    "status": "finished",
    "statusType": "error"
}
```

4.4.1.20 *PATCH*

Changes job status on the server. Can be used to cancel and resume Jobs.

Parameters:

none

Request Headers:

default (see 2.2.7)

Content-Type: *application/json*

Request JSON Object:

status: String, use *"cancelled"* to cancel jobs, and *" waitingForAccept"* to resume finished jobs, other values will be rejected.

Response Headers:

none

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

204 NoContent – on success

400 Bad Request

4.4.2 Ticket

Contains the complete Job Ticket (i.e. all the parameters actually used when starting job processing) for the given job. It is located at:

```
colorserver:<port>/jobs/:job-id/ticket
```

4.4.2.10 *GET*

Returns job's Ticket. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

clientId: ID of the client which started the job
jobCreator: display name of the client which started the job
inputFileId: resource ID of the input file
clientFilePath: storage path of the input file
creationTime: time when the job arrived at the Server (RFC 1123 date)

jobParams: template-specific parameters used with that job, either original template parameters or template parameters with overrides applied. This object contains JSON-serialized internal processing data used to start this job. Documentation of ist format is beyond the scope of this document!

jobTemplateIds: *Array* of types of the templates used with this job
jobTemplateNames: *Array* of names of the templates used with this job
jobTemplateTypes: *Array* template type IDs for the templates in both of the above arrays, ordering of all these three arrays must match!
subTemplatesTag: *Array* of pairs consisting of *templateId* and *templateTypeId*, its order is matching the order oft he previous three arrays. It is used internally to distinguish between vector and image templates when processing PDF files. You can ignore it.
jobTicketSupportedOperations: constraints for this job ticket, values:
deleteOnFinished – ticket has to be deleted after processing,
createOutput – the associated job is expected to create output files.
Used for internal bookkeeping, you can ignore it.

workerTypeId: type of worker processing this job (see 5.1.1)
localeId: the locale used with that job

Status Codes:

default (see 2.3.1)

Example:

```
{
    "clientFilePath": "C:/ /Hotfolder Test/IMG/input/Desert_CMYK_8bit_LZW.tif",
    "clientId": "HotfolderService@int.gmgcolor.com\\NBDETUSD28",
    "creationTime": "25-11-2016 07:54:03.145 GMT",
    "inputFileId": "{1a1ffc04-6ecb-49bd-ac7f-38c87903fc7b}",
    "jobCreator": "HotfolderService.Hotf IMG tst@int.gmgcolor.com\\NBDETUSD28",
    "jobParams": {
        "parameters": [ ... ]
    },
    "jobTemplateIds": [
        "{7002a9b4-191c-4b17-829b-2249e4137326}",
        "{f70f34be-d552-454d-bdd7-566ef51b58e4}"
    ],
    "jobTemplateNames": [
        "All to JPEG",
        "PSOcoated v3 to PSO MFC"
    ],
    "jobTemplateTypes": [
        "ImageProcessing",
        "ColorProcessing"
    ],
    "jobTicketSupportedOperations": "deleteOnFinished",
    "localeId": "de_DE",
    "subTemplatesTag": [
        {
              "templateId": "ImageProcessing",
            "templateTypeId": "ImageProcessing"
        },
        {
            "templateId": "ColorProcessing",
            "templateTypeId": "ColorProcessing"
        }
    ],
    "workerTypeId": "ImageProcessing"
}
```

4.4.3 Result

Contains the result of the given job. It is located at

```
colorserver:<port>/jobs/:job-id/result
```

4.4.3.10 *GET*

Returns job's result data. If not specified differently, the default type of JSON attributes is *String*.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:

outputFiles: contains an array of result file objects with:
- *fileId*: denoting files resource ID,
- *type*: file type. *"result" or "info"*

Note: the *"info"* files are files additionaly generated during job processing, for example ink saving reports for individual PDF files (see 4.2.3)

processingLog: contains the *messages* array of job log message objects with:
- *category*: message type - *"none", "info", "warning", "error", "critical"*
- *text*: the actual job processing log message
- *… :* other, less important attributes

resultCategory:"none", "info", "warning", "error", "critical", here *info* denotes successfull processing

informationDescription: additional textual information about the job status, mostly error message

startTime: time when processing was started for that job (RFC 1123 date)

finishedTime: time when processing was finished for that job (RFC 1123 date)

Status Codes:

[200 OK] – Request completed successfully

[404 Not Found] – Not found

Example:

```
{
    "informationDescription": "",
    "outputFiles": [
        {
            "fileId": "{4d0f6d63-d026-4393-befb-5f3ccb970455}",
            "type": "result"
        }
    ],
    "processingLog": {
        "messages": [
            {
                "category": 1,
                "iD": 0,
                "source": "PdfJobProcessor (PDF Job Processing)",
                "text": "Successfully finished processing",
                "timestamp": "2016-11-23 09-25-49 021",
                "typeId": "DefaultMessage"
            },
            ...
        ]
    },
    "resultCategory": "info",
    "startTime": "23-11-2016 08:24:42.765 GMT",
    "finishedTime": "23-11-2016 08:25:59.278 GMT"
}
```

5 Workers

5.1 Collection

This collection provides a list of Workers connected to the Server. It is located at:

```
colorserver:<port>/workers
```

5.1.1 GET

Returns a list of available Workers. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

If-Modified-Since - used to poll for collection‘s changes (see 2.2.6)

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

Response JSON Object:

array of JSON objects with following attributes:

workerType: unique type of the worker, denotes the type of jobs which this worker can process.

Note: the type of job's main template will be the same as the type of the worker processing it!

name: short name of the worker

description: detailed description of the worker

extensions: array of input file extensions this worker supports

Status Codes:

default (see 2.3.1)

304 Not Modified – If the collection wasn't modified since the time provided in the *If-Modified-Since* header

Example:

```
[
    {
        "workerType": "ImageProcessing",
        "name": "Image Color Conversion",
        "description": "",
        "extensions": [ "jpeg", "jpg", "tiff", "tif", ]
    },
    {
        "workerType": " PdfProcessing ",
        "name": " Pdf Color Conversion",
        "description": "",
        "extensions": [ "pdf", "jpeg", "jpg", "tiff", "tif" ]
    }
]
```

5.2 Elements

This collection doesn't have addressable elements - only the list of Workers can be requested.

6 Files

6.1 Collection

This collection provides a list of all files (for both inputs and results) either available physically on the Server (uploaded files), or known tot he server by their path only („linked" files). It is located at:

```
colorserver:<port>/files
```

As currently implemented, this collection supports only the „eventually consistent" mode! This means, that some internally created files (for example job output files, etc.) might initially not show up in the collection's list. However, when referenced by their IDs these files are always visible in the API.

Note: Not all versions and configurations of ColorServer 5 will support „linked" files as job input! Please consult the appropriate User Guide document to find out if your installation supports it.

6.1.1 GET

Returns a list of all files stored on the server. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

style [summary | detailed]: default is *summary*, if *detailed* used the representation will be that of 6.2.1.1

Request Headers:

default (see 2.2.7)

If-Modified-Since - used to poll for collection's changes (see 2.2.6)

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

Response JSON Object:

array of JSON objects with following attributes:

description: detailed description of the file

id: file's resource ID (unique, generated by the server or supplied by user)

name: file name on the server

linked: if false, this file was physically uploaded to the Server, if not, if is accessed by ist path, i.e. only a link is present on the server (*Boolean)*

Status Codes:

default (see 2.3.1)

304 Not Modified – If the collection wasn't modified since the time provided in the *If-Modified-Since* header

Example:

```
[
    {
        "description": "",
        "id": "{9b055cfe-ea38-4888-9413-e0e248529717}",
        "linked": true,
        "name": "Koala.jpg"
    },
    {
        "description": "Processing output",
        "id": "{622545ff-1799-4a19-aa50-9493b8b9549f}",
        "linked": false,
        "name": "Output_{453de464-b403-45e3-b0f9-0b574972046e} - Koala.jpg"
    },
]
```

6.1.2 POST

Registers a file with the server, returns a resource ID. This operation creates a „linked" file.

For this operation, the file path must be accessible from the server. This functionality is provided as to enable consistent addressing of file resources by unique resource IDs, in case the file upload is not required because the file can be directly accessed from the server.

Parameters: are to be included as request parameters in the URL

the ID to be used for the registered file– optional, has to be appended to the resource path

filePath – path of the file to be uploaded, must be accessible form the server!

clientID – ID of the client registering the file

Request Headers:

default (see 2.2.7)

Response Headers:

Location: path to the newly created resource, contains the resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully

400 Bad Request – Bad request

422 Unprocessable Entity – Server cannot access the given file

6.1.3 PUT

Uploads a file to the server.

Parameters: are to be included as request parameters in the URL

the ID to be used for the uploaded file– optional, has to be appended to the resource path (see Example 2)

fileName – name of the file to be uploaded, needed for the file suffix!

clientID – ID of the client uplaoding the file

Request Headers:

Content-Type: *application/octet-stream*

Request Body:

Binary profile file contents

Response Headers:

Location: path to the newly created resource, contains the resource ID

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully

400 Bad Request – Bad request

Example 1:

```
PUT  http://colorserver:80/files?fileName=inputFile.tiff &clientId=myUserId
Content-Type: application/octet-stream

<... File-body ...>
```

Response:

```
HTTP/1.1  201 Created
Server: Microsoft-HTTPAPI/2.0
Location: http://colorserver:80/files/{9f1f9366-2e1b-4d73-b92f-73d32de76f1d}
```

Example 2:

```
PUT  http://colorserver:80/files/{f93669f1-2e1b-4d73-b92f-de76f1d73d32}?fileName=
inputFile.tiff&clientId=myUserId
Content-Type: application/octet-stream

<... File-body ...>
```

Response:

```
HTTP/1.1  201 Created
Server: Microsoft-HTTPAPI/2.0
Location: http://colorserver:80/files/{f93669f1-2e1b-4d73-b92f- de76f1d73d32}
```

6.2 Elements

Single files available on the Server are adressed by their ID, e.g.:

```
colorserver:<port>/files/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
```

6.2.1 GET

The GET request can either fetch file's metadata or the entire input/output file, depending on the requested content type.

6.2.1.10 *Metadata*

Returns the detailed representation of a given color profile. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)
Accept: *application/json*

Response Headers:

Content-Type: *application/json*

Response JSON Object:

id: file's resource ID (unique, generated by the Server or supplied by user)
name: name of the file (including extension)
description: description of the file (generated by the Server)
storagePath: the full storage path on the server side. Depending on configuration, that this path can be direclty accessible from the client!

linked: if false, this file was physically uploaded to the Server, if not, if is accessed by ist path, i.e. only a link is present on the server (*Boolean*)

Status Codes:
default (see 2.3.1)

6.2.1.20 *File download*

Returns the file contents for a given server file (input or output).

Parameters:
none

Request Headers:
default (see 2.2.7)
Accept: *application/octet-stream*

Response Headers:
Content-Type: *application/octet-stream*

Response Body:
Binary file contents.

Status Codes:
default (see 2.3.1)

6.2.2 DELETE

Deletes an input/output file on the server. It should be always possible to delete a file.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
none

Response JSON Object:
none

Status Codes:
default (see 2.3.1)

7 Licenses

7.1 Collection

This collection provides a list of all licenses available on the Server. It is located at:

```
colorserver:<port>/licenses
```

Currently following license types are defined:

- •0 *serverBase:* needed for the Server to be started
- •0 *webApi:* enables Open REST-API
- •0 *colorServer:* enables full ColorServer configuration
- •0 *inkOptimizer:* enables „*InkOptimizer*“-only ColorServer configuration
- •0 *smartProfiler:* adds *SmartProfiler* product‘s functionality
- •0 *queuedJobs:* decides how many jobs can be run concurrently
- •0 *profiles:* enables usage of specific MX profiles

7.1.1 GET

Returns a list of all licenses configured on the server. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

If-Modified-Since: used to poll for collection‘s changes (see 2.2.6), only licenced-unlicenced changes will be considered, changes in the internal ID list are not taken into account at the moment!

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

Response JSON Object:

array of JSON objects with following attributes:

ids: array of internal licence IDs

name: licence name, one of: *serverBase, webApi, colorServer, inkOptimizer, smartProfiler, queuedJobs*

count: optionally, if the licence type supports counts, the count value will be included (*Number)*

Status Codes:

default (see 2.3.1)

304 Not Modified – If the any of the subcollections and any of theirs elements wasn’t modified since the time provided in the *If-Modified-Since* header

Note: You can query the types of MX profiles which are currently licensed on the Server by examing the iternal ID list of the *profiles* license!

Example:

```
[
    {
        "ids": [ 15, 10800 ],
        "name": "serverBase"
    },
    {
        "ids": [],
        "name": "colorServer"
    },
    {
        "count": 8,
        "ids": [ 10802 ],
        "name": "queuedJobs"
    }, ...
]
```

7.2 Subcollections

7.2.1 License Types Information

The *"types"* subcollection lets the user to get a list of all queryable licence types.

It is located at:

```
colorserver:<port>/licenses/types
```

7.2.1.10 *GET*

Returns a list of valid license names. If not specified differently, the default type of JSON attributes is *String*.

Parameters: none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:
array of client IDs, as supplied by users when starting jobs (see 4.1.2)

Status Codes:
default (see 2.3.1)

Example:

```
    [
        "serverBase",
        "webApi",
        "colorServer",
        "inkOptimizer",
        "smartProfiler",
        "queuedJobs",
        "profiles"
    ]
```

7.2.1.20 *Elements*

This collection doesn't have addressable elements - only the list of licence types can be requested.

7.3 Elements

Single Server licences are adressed by their names, e.g.:

```
colorserver:<port>/licenses/queuedJobs
```

7.3.1 GET

The GET request can be used in order to test if a given license is enabled on the Server.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:
ids: array of internal licence IDs
name: name of the license

Status Codes:
200 OK – License available on Server
404 Not Found – License not available on Server

Example 1:

```
GET /licenses/serverBase HTTP/1.1
```

Response:

```
{
    "ids": [ 15, 10800 ],
    "name": "serverBase"
}
```

Example 2:

```
GET /licenses/webApi HTTP/1.1
```

Response:

```
HTTP/1.1 404 Not Found
```

8 Resources

Resources are an umbrella term for different kinds of files which can be uploaded to the Color Server and subsequently used to parameterize job processing.

8.1 Resources Root

This collection provides a list of all resource files deployed on the Server. In general we have following groups of resources:

0

- •0 templates,
- •0 profiles,
- •0 spotcolor resources,
- •0 ICC substitutions,
- •0 workflows,
- •0 locations,
- •0 printers and
- •0 smart profiler resources.

All of these resource groups (i.e. subcollections of this collection) share common interface, in some cases exhibiting minor extensions and/or limitations.

This collection is located at

```
colorserver:<port>/resources.
```

It doesn't have addressable elements - only subcollections are present.

8.1.1.10 ***GET***

Returns a list of all resource instances deployed on the Server. This enables users to get single, comprehensive list of all resources acroos all resource types present on the Server. Moreover, the entire resource tree can be monitored for changes using this URI.

If not specified differently, the default type of JSON attributes is *String*.

Parameters:

range= start-[stop]: range of the resulting resource records to be returned. The parameter can be used instead of the *Range* header for ease of access. See below for meaning of *range* specifiers.

Request Headers:

default (see 2.2.7)

If-Modified-Since - poll for all changes of all subresource collections (see 2.2.6)

Range : *items=start-[stop]*

you can specify the range of resources in „items“, where the first item has the index of *0*. If the „stop“ index is ommited, the last item is assumed. Examples: *items=0-5, items=1-, items=5-10*

Response Headers:

Content-Type: *application/json*

Range : *items=<from>-<to>/** - returned if a specific range was requested

Accept-Ranges : *items* - returned if no specific range was requested

Last-Modified: last modification date for the collection

Response JSON Object:

array of JSON objects with following attributes:

href: relative URI of the resource

name: short name of the resource

type: textual description of the resource type

updated: timestamp of the last update

Status Codes:

default (see 2.3.1)

304 Not Modified – If the any of the subcollections and any of theirs elements wasn't modified since the time provided in the *If-Modified-Since* header

Example:

```
  [
     {
         "href": "resources/templates/image/{6ee231f0-b406-46ce-85cb-
a5a0da5bfe9a}",
         "name": "Image Invert Template",
         "type": "Image Template"
         "locked": false,
        "updated": "Tue, 16 Sep 2014 15:57:57 GMT"
     },
     {
         "href": "resources/templates/normalization/{c2dda196-400f-4320-9dca-
a6ed49ccde4f}",
         "name": "Default Normalization Template",
         "type": "Normalization Template"
         "locked": true,
        "updated": "Tue, 17 Sep 2014 16:03:57 GMT"
     },
     {
         "href": "resources/profiles/mx/{77ca8f9a-23b3-4d2a-910d-c9b64b3b748d}",
         "locked": false,
         "name": "CMYK2CMYKO",
         "type": "MX Profile",
         "updated": "Fri, 29 Aug 2014 13:02:48 GMT"
     },
     {
         "href": "resources/profiles/icc/{e2c6bc5c-9503-4a1f-957f-7c572b334713}",
         "locked": false,
         "name": "RSWOP",
         "type": "ICC Profile",
         "updated": "Fri, 29 Aug 2014 13:02:58 GMT"
     },
     {
         "href": "resources/spotcolor/db3/{68643ed7-547f-4210-9fad-27dba83cb5bc}",
         "locked": false,
         "name": "Simple.cdbx",
         "type": "Spotcolor Database",
         "updated": "Fri, 29 Aug 2014 13:03:51 GMT"
     },
     {
         "href": "resources/substitutions/{a5cfb39f-49d0-4c93-bdd5-7163c8e9fc3c}",
         "locked": false,
         "name": "test ICC substitution",
         "type": "ICC Substitution",
         "updated": "Mon, 13 Jul 2015 15:16:39 GMT"
     }
 ]
```

8.2 Generic Resources Interface

The resource root collection contains several resource types as subcollection which will be described in subsequent chapters. However, all of these resource types allow some common operations and their API follows some common structure.

In order to simplify this document, those generic operations will be described here. In case some resource types differ in behavior, this will be documented in the resource specific section.

8.2.1 Manipulation of Resources

Physically, resources are <u>always saved as files</u>, where different resource types use different file extensions.

On the server side new resource instances are created by uploading resource files (*POST*). Existing resource instances can be changed by replacing the old contents with a newly uploaded resource file (*PUT*). Some simple metainfo attributes can be changed without uploading of an entire changed resource file (*PATCH*).

Resources can be locked by setting their *"locked"* attribute (*PATCH*). At last, resource files can be downloaded to be further edited locally (*GET*), their metainfo can be queried (*GET*), and they can be permanently removed from server (*DELETE*).

Note: The detailed formats of different resource files is beyond the scope of this specification and are due to change! The user should create and edit resources using the appropriate GUI of the ColorServer product.

8.2.2 Resource Type Collection

This collection provides a list of all resources of given type available on the Server. It is located at:

```
colorserver:<port>/resources/:resource-type
```

Moreover, resources of a given type can be subdivided in subcollections that respectively contain resources of different subtypes (see 8.2.3). For example under the *"profiles"* resource path there are two profile subtypes: *"profiles/icc"* and *"profiles/mx"*.

8.2.2.10 *GET*

Returns a list of all resources of given type deployed on the Server. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

style [summary | detailed]: default is *summary*, if *detailed* is used the representation will be that of 8.2.4.1.1, with contents of linked subresources being included

Request Headers:

default (see 2.2.7)

If-Modified-Since - used to poll for collection‘s changes (see 2.2.6)

If-None-Match: also used to poll for collection’s changes. It does not rely on RFC 1123 date precision as it uses ETags (see 2.2.3).

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

ETag: the entity tag for the collection (i.e. hash of states of all contained resource files)

Response JSON Object:

array of JSON objects with following attributes:

href: relative URI of the resource

name: short name of the resource

type: resource type or subtype string, e.g. for *templates* it will be "Image Template", "PDF Template", etc.
updated: timestamp of the last update
locked: resource's lock state, true/false (*Boolean*)

Status Codes:

default (see 2.3.1)
304 Not Modified – If the collection and any of ist elements wasn't modified since the time provided in the *If-Modified-Since* header, or if the current entity tag does not differ from the value of *If-None-Match* header

Example 1: Summary templates representation

```
[
    {
        "href": "resources/templates/image/{92d3991d-fe90-415c-8044-6a4a34344e3}",
        "locked": false,
        "name": "Default test template IMG-Proc",
        "type": "Image Template",
        "updated": "Wed, 04 Nov 2015 08:36:08 GMT"
    },
    {
        "href": "resources/templates/pdf/{0d1f070f-71ce-4378-b016-056e3cb0fa58}",
        "locked": false,
        "name": "Default test template PDF-Proc",
        "type": "PDF Template",
        "updated": "Wed, 04 Nov 2015 08:36:09 GMT"
    },
    {
        "href": "resources/templates/color/{6c9206c7-4dbf-498f-962e-bfde73cd2a2}",
        "locked": false,
        "name": "Default test template COLOR-Mgm",
        "type": "Color Template",
        "updated": "Wed, 04 Nov 2015 08:37:32 GMT"
    }
]
```

Example 2: Detailed templates representation

```
 [
    {
        "fileSizeKb": 2.89,
        "href": "resources/templates/normalization/{9dff4d30-4d86-4a88-8dc7-
20c9141e437c}",
        "license": "ok",
        "locked": false,
        "name": "Default test template Normalization",
        "parameters": {
            "checksum": "2967538579",
            "parameters": [
                {
                    "id": "ColorSpaceNormalization",
                    "value": {
                        "cmykNormalizationRules": { ... },
                        ...
                    }
                }
            ],
            "updated": "Thu, 27 Oct 2016 08:37:13 GMT",
            "validationMessages": [],
            "validationState": "info"
        },
        "type": "Normalization Template",
        "updated": "Thu, 27 Oct 2016 08:37:13 GMT"
    },
    ...
]
```

8.2.2.20 *POST*

Uploads a resource file to the server, creates a new instance of given resource type.

Parameters: to be included as parameters in the URL

the ID to be used for the uploaded resource – optional, has to be appended to the resource path (see Example 2)

fileName – filename (with extension) of the new resource file.
The resource subtype will be inferred from the file extension, and the resource will be automatically placed in the right subcollection!
The name of the resource will be derived from this value by discarding the file type extension.

description – description for the new resource (optional)

Request Headers:

Content-Type: *application/octet-stream*

Request Body:

Binary resource file contents

Response Headers:

Location: path to the newly created resource, contains the resource ID
ETag: the entity tag (i.e. hash of ressource's current content)

Response JSON Object:

in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:

201 Created – Request completed successfully
400 Bad Request – Bad request

Example 1:

```
  POST  http://colorserver:80/resources/templates?fileName=ImageTemplate.xipt
&description=Some%20New%20Template
  Content-Type: application/octet-stream

  <... File-body ...>
```

Response:

```
  201 Created
  Location: http://colorserver:80/resources/templates/image/{9f1f9366-2e1b-4d73-
  b92f-73d32de76f1d}
```

Example 2:

```
POST  http://colorserver:80/resources/templates/{2e1f9366-fd1b-4d73-b92f-
32de76f1d73d}?fileName=PdfTemplate.xppt&description=Some%20Newer%20Template
Content-Type: application/octet-stream

<... File-body ...>
```

Response:

```
201 Created
Location: http://colorserver:80/resources/templates/pdf/{2e1f9366-fd1b-4d73-
b92f-32de76f1d73d}
```

8.2.3 Subcollections

The resource path for a given resource type can contain subcollections grouping resources of a given subtype.

Note: In case of resources without subtypes, this API will be used for such resource collections instead of the more general 8.2.2!

8.2.3.10 *Resource Subtype Collections*

These subcollections correspond to the resource subtypes defined for the given resource type. They are correspondingly located at:

```
colorserver:<port>/resources/:resource-type/:resource-subtype
```

Where `:resource-subtype` can be e.g.: *"pdf", "image", "color"* and *"normalization"* for the `:resource-type` being *"templates"*.

8.2.3.1.10 GET

The resource subcollections can be queried in exactly the same way as the main resource collection, with the single difference, that only resources of given subtype will be affected.

However, the format of the returned data is slighltly changed:

Response JSON Object:
array of JSON objects with following attributes:
description: detailed description of the resource
id: profile's resource ID (unique, generated by the server or supplied by user)
name: short name of the profile
updated: timestamp of the last update
locked: true/false (*Boolean*)

i.e. the *href* parameter of 8.2.2.1 gets replaced by the *id*.

Example:

```
[
    {
        "description": "new description, EDITED (from TESTS....)",
        "id": "{47c9bf42-9223-4b74-9fca-7bc8d25447c8}",
        "locked": false,
        "name": "Default test template IMG-Proc",
        "updated": "Thu, 05 Nov 2015 08:38:56 GMT"
```

```
      },
    ...
  ]
```

8.2.3.1.20 POST

The POST requests may se sent directly to the resource-type subcollection in the same manner as to the main collection. However in this case the file extension has to match the type of the subcollection! If that's not the case:

Response JSON Object:
in error case: error response with following attributes,
message - *File extension doesn't match repository type.*

8.2.4 Elements

Single resources present at the Server are adressed in their appropriate subcollections by their IDs or names, e.g.:

```
colorserver:<port>/resources/:resource-type/:resource-subttype/:resource-id
colorserver:<port>/resources/:resource-type/:resource-subttype/:resource-name
```

This means, that inside of resource subtypest he resource names have to be unique!

However, because the resource ID is guaranteed to be unique too, each resource can be addressed from the top-level resource type collection too, e.g.:

```
colorserver:<port>/resources/:resource-type/:resource-id
```

Some resources may require a separate licence tob e present at the server. If a licence is not found then no operations will be allowed, except for reading of resource's metainfo.

8.2.4.10 *GET*

The GET request can either fetch resource's metadata or the entire resource file, depending on the requested content type.

8.2.4.1.10 Metadata

Returns the detailed information about a given resource. If not specified differently, the default type of JSON attributes is *String*.

Parameters:
style [summary | *detailed]:* default is *summary*, if *detailed* used the contents of subresources will be included

Request Headers:
default (see 2.2.7)
Accept: *application/json*
If-None-Match: client's current entity tag, used to check for resource update

Response Headers:
Content-Type: *application/json*
ETag: the entity tag (i.e. hash of ressource's current content)

Response JSON Object:

id: resource's resource ID (unique, generated by the server or supplied by user)
name: short name of the resource
description: detailed description of the resource
fileName: the original name of the file used to create this resource
fileSizeKb: size (in kBytes) of the resource file (*Number*)
locked: the lock state oft he resource, true/false *(Boolean)*
license: *none* - not needed for the resource, *ok* - resource correctly licenced, *invalid* - resource not licenced (see also 2.2.1)
updated: timestamp of the last update
parameters: link to the subresource, if *detailed* used, see 8.2.5.1.1.

Status Codes:

default (see 2.3.1)
304 Not Modified – If the resource's current Etag matches that provided in the *If-None-Match* header

Example:

```
{
    "description": "new description, EDITED",
    "fileName": "Test template IMG-Proc.xipt",
    "fileSizeKb": 1.2,
    "id": "{47c9bf42-9223-4b74-9fca-7bc8d25447c8}",
    "locked": false,
    "license": "ok",
    "name": "Default test template IMG-Proc",
    "parameters": {
        "href": "/parameters"
    },
    "updated": "Thu, 05 Nov 2015 08:38:56 GMT"
}
```

8.2.4.1.20 File download

Returns the contents for a given resource file.

Parameters:

none

Request Headers:

default (see 2.2.7)
Accept: *application/octet-stream*
If-None-Match: client's current entity tag, used to check for resource update

Response Headers:

Content-Type: *application/octet-stream*
ETag: the entity tag (i.e. hash of ressource's current content)

Response Body:

Binary resource file contents.

Status Codes:

default (see 2.3.1)

304 Not Modified – If the resources's current Etag matches that provided in the *If-None-Match* header

8.2.4.20 *DELETE*

Deletes a resource on the server. Note that a locked resource cannot be deleted and the request will be rejected with *400 Bad Request* in such a case.

Parameters:
none

Request Headers:
default (see 2.2.7)
If-Match: client's current entity tag, used to check for third-party modifications

Response Headers:
none

Response JSON Object:
in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:
204 NoContent – on success
400 Bad Request
412 Precondition Failed – if ETag used, the resource was modified by another client

8.2.4.30 *PATCH*

Changes attribute of a resource on the server. Note that a locked resource cannot be patched.

Parameters:
none

Request Headers:
Content-Type: *application/json*
If-Match: client's current entity tag, used to check for third-party modifications

Request JSON Object: note that per request only one attribute can be set!
name: new name for the color profile (*String*)
description: new description for the color profile (*String*)
locked: use true to lock, false to unlock the resource (*Boolean*)

Response Headers:
ETag: the entity tag (i.e. hash of ressource's current content)

Response JSON Object:
in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:
204 NoContent – on success
400 Bad Request

412 Precondition Failed – if ETag used, the resource was modified by another client

Example:

```
PATCH /resources/templates/{2bced842-fcf7-46c4-ba24-b373444e664b} HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
    "locked": true
}
```

8.2.4.40 *PUT*

Replaces the contents of an existing resource with that of a given binary file. Note that a locked resource's contents cannot be replaced.

Parameters: to be included as parameters in the URL
fileName – filename (with extension) of the new resource file. The existing resource's type will checked against the file extension.

Request Headers:
Content-Type: *application/octet-stream*
If-Match: client's current entity tag, used to check for third-party modifications

Request Body:
Binary resource file contents

Response Headers:
ETag: the entity tag (i.e. hash of ressource's current content)

Response JSON Object:
in case of *Bad Request*: the error description (see 2.3.2), else *none*

Status Codes:
200 OK – Request completed successfully
400 Bad Request – Bad request
412 Precondition Failed – if ETag used, the resource was modified by another client

Example:

```
  PUT /resources/templates/{2bced842-fcf7-46c4-ba24-b373444e664b}}?fileName=Pdf
Resource.xppt HTTP/1.1
  Content-Type: application/octet-stream
  If-Match: "xyz-zzzxxx"

  <... File-body ...>
```

8.2.5 Subresources

Every resource instance contains at least one subresource named "*parameters*". Some resource types define additional subresources.

8.2.5.10 *Parameters*

This subresource contains some general, file-related metainfo, the current validation state, and, optionally, additional contents extracted from the physical resource file uploaded to server.

It is located at:

```
colorserver:<port>/resources/:resource-type/:subtype/:resource-name/parameters
colorserver:<port>/resources/:resource-type/:subtype/:resource-id/parameters
colorserver:<port>/resources/:resource-type/:resource-id/parameters
```

8.2.5.1.10 GET

Returns template's processing parameters.

Parameters:
none

Request Headers:
default (see 2.2.7)

Response Headers:
Content-Type: *application/json*

Response JSON Object:
checksum: the checksum contained in the given resource file
updated: timestamp of the last update of the given resource

validationState: contains validation result for this resource instance, values are *"none"*, *"info"*, *"warning"*, *"error"*, *"critical"*, here *"info"* denotes a successful validation

validationMessages: an *Array* of validation message *String*s, may be empty.

parameters: optional, contains an *Array* of resource specific parameter value objects in the general form of:

```
{
  "id": "setting-group-id",
  "value": { … }  // settings, resource type specific
}
```

This corresponds to the JSON-serialized contents of the physical resource file uploaded to server. **Note**: this attribute is not suported by all resource types!

Status Codes:
default (see 2.3.1)

Example: Parameters of an example image processing template

```
{
    "checksum": "2967538579",
    "parameters": [
        {
            "id": "ColorSpaceNormalization",
            "value": {
                "cmykNormalizationRules": { ... },
                ...
            }
        }
    ],
    "updated": "Thu, 27 Oct 2016 08:37:13 GMT",
    "validationMessages": [],
    "validationState": "info"
}
```

Note: The detailed formats of processing parameters for different resource types is beyond the scope of this specification and are due to change!

8.3 Templates

Templates are saved, preconfigured job tickets that can be applied to a given input file in order to start a color transformation job. There are four types of templates: image templates, PDF templates, color conversion templates and normalization templates.

8.3.1 Collection

This collection provides a list of all templates available on the Server. It is located at:

```
colorserver:<port>/resources/templates
```

It implements the generic resource collection interface as described in 8.2.2, i.e.

- GET for listing of templates and
- POST for creating new template instances.

8.3.2 Subcollections

8.3.2.10 *Template Type Subcollections*

The templates resource path contains subcollections grouping templates of a given type. They are correspondingly located at:

```
colorserver:<port>/resources/templates/image
colorserver:<port>/resources/templates/pdf
colorserver:<port>/resources/templates/color
colorserver:<port>/resources/templates/normalization
```

The allowed file extensions for different template types used in PUT and POST requests are:

Image templates:	*.xipt*
PDF templates:	*.xppt*
Color templates:	*.xcpt*
Normalization templates:	*.xcnt*

These subcollections implement generic resource subtype collection interface as described in 8.2.3.1, i.e.

- GET for retrieving the list of subtype collection's elements and
- POST for creating new template instances.

8.3.2.20 *Template Type Information*

Additionally, the "*types*" subcollection enables users to query the internal IDs of the supported template types. It is located at:

```
colorserver:<port>/resources/templates/types
```

8.3.2.2.10 GET

Returns a list of type IDs for all templates types currently available on the Server. If not specified differently, the default type of JSON attributes is *String*.

Parameters: none

Request Headers:

default (see 2.2.7)

If-Modified-Since - used to poll for collection's changes (see 2.2.6)

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

Response JSON Object:

array of JSON objects with following attributes:

id: ID of the template type

name: description of the template type

href: relative URI of the subcollection containing this template type

Status Codes:

default (see 2.3.1)

304 Not Modified – If the collection and any of ist elements wasn't modified since the time provided in the *If-Modified-Since* header

Example:

```
[
    {
        "id": "ColorNormalization",
        "name": "Normalization Templates",
        "href": "resources/templates/normalization"
    },
    {
        "id": "ColorProcessing",
        "name": "Color Templates",
        "href": "resources/templates/color"
    },
    {
        "id": "ImageProcessing",
        "name": "Image Templates",
        "href": "resources/templates/image"
    },
    {
        "id": "PdfProcessing",
        "name": "PDF Templates",
        "href": "resources/templates/pdf"
    }
]
```

8.3.3 Elements

Single templates present at the Server are adressed in their appropriate subcollections by their IDs or names, or directly from the top-level *templates* collection by their ID, according to generic resource collection interface:

```
colorserver:<port>/resources/templates/pdf/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
colorserver:<port>/resources/templates/image/Example%20Image%20Template

colorserver:<port>/resources/templates/{6ee231f0-b406-46ce-85cb-a5a0da5bfe9a}
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- GET for retrieving of template metainfo or its resource file contents,
- DELETE for removing of template instances,
- PATCH for changing of template metainfo attributes and
- PUT for replacing of entire underlying template resource files.

8.3.4 Subresources

Every instance contains a single *"parameters"* subresource containing template's processing parameters. It implements the generic resource *parameters* subresource interface, as described in 8.2.5.1, i.e.

- GET for retrieving of template processing parameters.

Example:

```
GET /resources/templates/image/{7002a9b4-191c-4b17-829b-2249e4137326}/parameters
```

Response:

```
{
  "checksum": " 3323852349",
  "parameters": [
          "id": "OutputFileFormat",
          "value": {
              "bitDepth": "default",
              "encodingType": "lzw",
              "jpegQuality": "high",
              "keepCompression": true,
              "keepEncoding": true,
              "outputFileFormat": "jpeg"
          }
      },
      {
          "id": "Resampling",
          "value": {
              "interpolationMode": "noInterpolation",
              "outputResolution": 300,
              "resolutionRange": {
                  "maxValue": 4826,
                  "minValue": 2.54
              },
              "resolutionRangeEnabled": false
          }
      },
      {
          "id": "Sharpening",
          "value": {
              "sharpeningEnabled": false,
              "sharpeningLevel": 0
          }
      }
```

```
    ],
    "updated": " Wed, 05 Oct 2016 20:03:04 GMT",
    "validationMessages": [],
    "validationState": "none"
  }
```

Note: The detailed formats of processing parameters for different resource types is beyond the scope of this specification and are due to change!

8.4 Profiles

Profiles are resources used for direct color transformations. They are subdivided in two subcollections containing respectively the pripríetary GMG MX profiles and the standard ICC ones.

8.4.1 Collection

This collection provides a list of all color profiles available on the Server. It is located at:

```
colorserver:<port>/resources/profiles
```

It implements the generic resource collection interface as described in 8.2.2, i.e.

- •0 GET for listing of templates and
- •0 POST for creating new template instances,

with some extensions which are described in the following section.

8.4.1.10 *GET*

In addition to the generic resource interface as decribed in 8.2.2.1 this collection defines additional parameters to enable querying for specific profiles.

Parameters:

colorSpace - filters out both ICC and MX profiles with input or output color spaces equal to the specified value, allowed values: *cmyk, rgb, lab, gray, multicolor*

8.4.2 Subcollections

8.4.2.10 *Profile Type Subcollections*

The profiles resource path contains subcollections corresponding to the profile types defined in the system, i.e. ICC and MX. They are correspondingly located at:

```
colorserver:<port>/resources/profiles/mx
colorserver:<port>/resources/profiles/icc
```

The allowed file extensions for different profile types used in PUT and POST requests are:

.icc for ICC profiles, and
.mx3, .mx3x, .mx4, .mx4x for MX profiles.

These subcollections implement generic resource subtype collection interface as described in 8.2.3.1, i.e.

- •0 GET for retrieving the list of subtype collection's elements and
- •0 POST for creating new template instances.

with some extensions which are described in the following section.

8.4.2.20 *GET*

In addition to the generic resource interface as decribed in 8.2.3.1.1 this collection defines additional parameters to enable querying for specific profiles

Parameters:

MX subpath only:

profileType - filters out specific profile types, values: *calibration, proof, conversion, separation, inkOptimizer*

profileFormat - filters out specific profile formats, values: *mx3*, *mx4*

colorSpace - filters out profiles with specific color spaces, values: *cmyk, rgb, multicolor*

ICC subpath only:

profileType - filters out specific profile types, values: *pcs, devLink*

profileClass - filters out specific profile classes, values: *display, output*

colorSpace - filters out profiles with specific color spaces, values: *cmyk, rgb, lab, grayscale*

8.4.3 Elements

Single profiles present at the Server are adressed in their appropriate subcollections by their IDs or names, or directly from from the top-level *profiles* collection by their ID, according to generic resource collection interface:

```
colorserver:<port>/resources/profiles/icc/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
colorserver:<port>/resources/profiles/mx/Some%20MX%20Profile
colorserver:<port>/resources/profiles/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- •0 GET for retrieving of profile metainfo or its resource file contents,
- •0 DELETE for removing of profile instances,
- •0 PATCH for changing of profile metainfo attributes and
- •0 PUT for replacing of entire underlying profile resource files.

8.4.4 Subresources

Every instance contains a single "*parameters*" subresource containing basic profile information. It implements the generic resource instance's "*parameters*" subresource interface, as described in 8.2.5.1, i.e.

- •0 GET for retrieving of template processing parameters,

with some differences which are described in the following section.

8.4.4.10 *Parameters*

8.4.4.1.10 GET

Profiles exhibit a slightly different format for the "*parameters*" subresource representation than the other resources implementing the generic interface from 8.2.5.1. If not specified differently, the default type of JSON attributes is *String*.

Response JSON Object:

profile type specific parameters, containing:

updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1
inputColorSpace: CMYK, RGB, multicolor, LAB, gray
outputColorSpace: CMYK, RGB, multicolor, LAB, gray
checksum: the checksum contained in the profile file
colorChecksum: additional checksum computed only for the data relevant to color processing!

MX specific parameters:

profileFormat: file extension for that profile file (MX*3, MX4, MX)*
p*rofileType*: description of MX profile type. There are following general MX profile type groups: *Calibration, Proofing, Conversion, Separation, Ink Optimizer*. The type decription string will contain this type name plus some additional information.
Examples: *"RGB2CMYK Conversion", "CMYK2CMYK Conversion", "CMYK Proofing", "Ink Optimizer Gravure"*.
profileLicences: an *Array* of licence IDs (*String*) for this MX profile
calibration: can this profile be used for calibration? (*Boolean*)
inputColorSpaceDescription: text describing additional information about the input color space
outputColorSpaceDescription: text describing additional information about the output color space
profileIntentDescription: additional information about profile usage

ICC specific parameters:

p*rofileType*: *Output device, Display, Device link*

Example 1: Parameters of an example MX profile

```
{
    "calibration": false,
    "checksum": "722a6207",
    "colorChecksum": "",
    "inputColorSpace": "RGB",
    "inputColorSpaceDescription": "",
    "outputColorSpace": "CMYK",
    "outputColorSpaceDescription": "SNAP 2007",
    "profileFormat": "MX4",
    "profileIntentDescription": "Coldset-Offset Newsprint Paper",
    "profileLicences": [
        "8061"
    ],
    "profileType": "RGB2CMYK Conversion",
    "profileTypeDescription": "SeparateRGB",
    "updated": "Thu, 07 Jul 2016 15:02:59 GMT",
    "validationMessages": [],
    "validationState": "info"
}
```

Example 2: Parameters of an example ICC profile

```
{
    "checksum": "a4eb3236a03c2355a6db63c8fd7cfb85",
    "colorChecksum": "3219714557",
    "inputColorSpace": "CMYK",
    "outputColorSpace": "CMYK",
    "profileType": "Output device",
    "updated": "Thu, 07 Jul 2016 16:09:37 GMT",
    "validationMessages": [],
    "validationState": "info"
}
```

8.5 Spotcolor Resources

Spotcolor resources are color resources used in transformations of spotcolors.

8.5.1 Collection

This collection provides a list of all spotcolor resources available on the Server. The collection is located at:

```
colorserver:<port>/resources/spotcolor
```

It implements the generic resource collection interface as described in 8.2.2, i.e.

- •0 GET for listing of spotcolor resources and
- •0 POST for creating new spotcolor resource instances,

8.5.2 Subcollections

The spotcolor resource path contains subcollections corresponding to the spotcolor resource types defined in the system, i.e. DB3 databases, spotcolor rulestes and spotcolor gradations. They are located at:

```
colorserver:<port>/resources/spotcolors/db3
colorserver:<port>/resources/spotcolors/gradations
colorserver:<port>/resources/spotcolors/rulesets
```

The allowed file extensions for different spotcolor resource types used in PUT and POST requests are:

DB3 database files:	*.db3* and *.cdbx*
Gradation files:	*.sfg*
Rulesets files:	*.scc*

These subcollections implement generic resource subtype collection interface as described in 8.2.3.1, i.e.

- •0 GET for retrieving the list of subtype collection‘s elements and
- •0 POST for creating new spotcolor resource instances,

with some extensions which are described in the following section.

8.5.2.10 *GET*

In addition to the generic resource interface as decribed in 8.2.3.1.1 the DB3 subcollection defines parameters which enable users to query for DB3 databases containig specific colors or color sets.

Parameters:

colorSet=<set-name>: query only for color databases containing that color set

Note: for user convenience, if a color set contains the "®" character, this character can be omitted in filter, e.g. filter value *"PANTONE PLUS coated"* will select the color set name *"PANTONE® PLUS coated"*!

color=<color-name>: query only for color databases containing that color, '*' is allowed as wildcard

Examples: *color=PANTONE%20100%20C*
*color=PANTONE**
*color=DIC%209**

Note: if color lists are not supported in your ColorServer 5 installation then filtering by color will not be supported as well! See 8.5.4.1.1.1**Error! Reference source not found.**.

Example:

```
GET /resources/spotcolor/db3?colorSet=HKS_N*
```

Response:

```
[
   {
       "href": "resources/spotcolor/db3/{dc2463de-1e58-4d4a-83f6-1b26643d7977}",
       "locked": false,
       "name": "GRACoL2006coatedcommercialsheet1",
       "type": "Spotcolor Database",
       "updated": "Wed, 05 Oct 2016 20:03:04 GMT"
   },
   {
       "href": "resources/spotcolor/db3/{90a4b105-e135-4bac-990d-b1fc45e325fc}",
       "locked": false,
       "name": "ISOcoated27L",
       "type": "Spotcolor Database",
       "updated": "Wed, 05 Oct 2016 20:03:04 GMT"
   },
   {
       "href": "resources/spotcolor/db3/{f3429055-e117-4b1c-af3e-d9a48b526c43}",
       "locked": false,
       "name": "ISOcoatedv2-39L",
       "type": "Spotcolor Database",
       "updated": "Wed, 05 Oct 2016 20:03:04 GMT"
   },
   ...
]
```

8.5.3 Elements

Single spotcolor resources present at the Server are adressed in their appropriate subcollections by their IDs or names.

However, as all three subresources are playing totally different role in color processing, the acces over the common root path .i.e. `resources/spotcolors` is disabled!

```
colorserver:<port>/resources/spotcolors/db3/{9e3b5c2a-...-369dd2c32522}
colorserver:<port>/resources/spotcolors/db3/Simple%20cdbx
```

```
colorserver:<port>/resources/spotcolor/gradations/{9e3b5c2a-...-369dd2c32522}
colorserver:<port>/resources/spotcolor/gradations/Sample%20Gradation%20File

colorserver:<port>/resources/spotcolor/rulesets/{9e3b5c2a-...-369dd2c32522}
colorserver:<port>/resources/spotcolor/rulesets/Default%20Ruleset%File
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- •0 GET for retrieving of spotcolor resource metainfo or its resource file contents,
- •0 DELETE for removing of spotcolor resource instances,
- •0 PATCH for changing of spotcolor resource metainfo attributes and
- •0 PUT for replacing of entire underlying spotcolor resource resource files,

with some extensions which are described in the following sections.

8.5.3.10 *GET*

8.5.3.1.10 Metadata

Spotcolor Gradations and Rulesets provide the standard format for the element's metadata representation conforming with the generic interface from 8.2.4.1.1.

Spotcolor DB3 databases however exhibit a different format for the element's metadata representation than the other resources implementing the generic interface from 8.2.4.1.1.

Response JSON Object:

id, *name, parameters,* etc.: standard element attributes as in 8.2.4.1.1.

sets: link to the additional subresource, namely list of contained color sets, if *detailed* used this subresource won't be included, because it is already contained in the standard "*parameters*" subresource.

set-elemets: pseudo-link, HATEOAS-like description of how to access individual color set's data

Example 1:

```
  {
    "description": "",
    "fileName": "SpotColors_GRACoL2006coatedcommercialsheet1.cdbx",
    "fileSizeKb": 689.28,
    "id": "{dc2463de-1e58-4d4a-83f6-1b26643d7977}",
    "license": "ok",
    "locked": false,
    "name": "GRACoL2006coatedcommercialsheet1",
    "parameters": {
        "href": "/parameters"
    },
    "sets": {
        "href": "/sets"
    },
    "set-element": {
        "href": "/sets/set-id"
    },
    "updated": "Wed, 05 Oct 2016 20:03:04 GMT"
}
```

Example 2:

```
GET /resources/spotcolor/db3/Simple.cdbx?style=detailed HTTP/1.1
Accept: application/json
Host: colorserver:8011
```

Response:

```
  {
    "description": "",
    "fileName ": "Simple.cdbx.db3",
    "fileSizeKb": 32.37,
    "id": "{fb959bc8-0ca8-446c-b197-3ba790d8a70b}",
    "license": "ok",
    "locked": false,
    "name": "Simple.cdbx",
    "parameters": {
        "checksum": "1323766389",
        "colorSets": [
            "DIC Color Guide®",
            "HKS_K",
            "PANTONE® PLUS coated"
        ],
        "colorantCount": 1183,
        "colorants": [
            "DIC 100s [DIC Color Guide®] (L: 61.21 a: -22.79 b: -31.32)",
            "DIC 101s [DIC Color Guide®] (L: 63.52 a: -6.32 b: -38.33)",
            ...
            "HKS 1 K [HKS_K] (L: 89.1 a: 2.25 b: 60.28)",
            "HKS 11 K [HKS_K] (L: 72.2 a: 41.58 b: 39.38)",
            ...
            "PANTONE 100 C [PANTONE® PLUS coated] (L: 92.04 a: -7.56 b: 65.76)",
            "PANTONE 101 C [PANTONE® PLUS coated] (L: 91.76 a: -7.51 b: 75.11)",
        ],
        "updated": "Mon, 30 May 2016 16:55:28 GMT",
        "validationMessages": [],
        "validationState": "none"
    },
    "updated": "Mon, 30 May 2016 16:55:28 GMT"
}
```

8.5.4 Subresources

Every instance contains a single *"parameters"* subresource containing spotcolor resource information. It implements the generic resource instance's *"parameters"* subresource interface, as described in 8.2.5.1, i.e.

- •0 GET for retrieving of spotcolor resource's processing parameters,

with some differences which are described in the following section.

8.5.4.10 *Spotcolor DB3 Databases*

In case of the spotcolor subresource types, the DB3 subresource type exibits very different interface than the remaining two subresources.

8.5.4.1.10 Parameters

8.5.4.1.1.1 GET

Spotcolor databases exhibit a different format for the *"parameters"* subresource representation than the other resources implementing the generic interface from 8.2.5.1.

Response JSON Object:

checksum: as in 8.2.5.1.1

updated: as in 8.2.5.1.1

validationMessages: as in 8.2.5.1.1

validationState: as in 8.2.5.1.1

colorSets – array of names of contained color sets

colorants – all contained colors tagged with their color set and CMYK values,

Format: *"color-name [color-set] (CMYK-encoding)"*

Note: the color list <u>can be left empty</u> in some Color Server 5 versions and configurations!

colorantCount – number of all contained colors (*Number*)

Example:

```
  {
     "checksum": "1323766389",
     "colorSets": [
        "DIC Color Guide®",
        "HKS_K",
        "PANTONE® PLUS coated"
     ],
     "colorantCount": 1183,
     "colorants": [
        "DIC 100s [DIC Color Guide®] ( Cyan: 0.591 Magenta: 0.0933 Yellow: 0.0203
Black: 0.121 )",
        "DIC 101s [DIC Color Guide®] (Cyan: 0.5124 Magenta: 0.2377 Yellow: 0.0038
Black: 0.02)",
          ...
        "HKS 1 K [HKS_K] ( Cyan: 0 Magenta: 0.1171 Yellow: 0.645 Black: 0 )",
        "HKS 11 K [HKS_K] ( Cyan: 0 Magenta: 0.5807 Yellow: 0.5848 Black: 0 )",
          ...
        "PANTONE 100 C [PANTONE® PLUS coated] ( Cyan: 0 Magenta: 0.0194 Yellow:
0.6288 Black: 0 )",
        "PANTONE 101 C [PANTONE® PLUS coated] ( Cyan: 0 Magenta: 0.022 Yellow:
0.7118 Black: 0 )",
     ],
     "updated": "Mon, 30 May 2016 16:55:28 GMT",
     "validationMessages": [],
     "validationState": "none"
  }
```

8.5.4.1.20 Sets

Contains the list of color sets present in the given database. It is located at:

```
colorserver:<port>/resources/spotcolor/db3/:db3-id/sets
```

8.5.4.1.2.1 GET

Returns the list of color sets of the database.

Parameters:

none

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

Array of color set names.

Status Codes:

default (see 2.3.1)

Example:

```
[
  "DIC Color Guide®",
  "HKS_K",
  "PANTONE® PLUS coated"
]
```

8.5.4.1.2.2 Subresources

Each of the color set names returned in the above list names a subresource on ist own. Such a resource is located at:

```
colorserver:<port>/resources/spotcolor/db3/:db3-id/sets/:color-set-name
```

Note: For convenience's sake, if a color set name contains the "®" character, this character may be (optionally) omitted when specifying color set's name!

8.5.4.1.2.2.1 GET

Returns detailed information about a given color set contained in the database. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

name – name of the color set

id – internal unique ID of the color set

colorants – list of all colors in this color set tagged with their CMYK values,

Format: *"color-name (CMYK-encoding)"*

Note: the color list can be left empty in some ColorServer 5 versions and configurations!

Status Codes:
default (see 2.3.1)

Example:

```
  GET /resources/spotcolor/db3/Example.db3/sets/PANTONE%C2%AE%20PLUS%20coated
```
Or:
```
  GET /resources/spotcolor/db3/Example.db3/sets/PANTONE%20PLUS%20coated
```

Response:

```
{
  "colorants": [
     "PANTONE 100 C ( Cyan: 0 Magenta: 0.0194 Yellow: 0.6288 Black: 0 )",
     "PANTONE 101 C ( Cyan: 0 Magenta: 0.022 Yellow: 0.7118 Black: 0 )",
     "PANTONE 102 C ( Cyan: 0 Magenta: 0.0086 Yellow: 1 Black: 0 )",
     "PANTONE 103 C ( Cyan: 0.0307 Magenta: 0.1099 Yellow: 1 Black: 0.1692 )",
     "PANTONE 104 C ( Cyan: 0.0547 Magenta: 0.1326 Yellow: 0.9478 Black: 0.2456)",
     "PANTONE 105 C ( Cyan: 0.1027 Magenta: 0.1983 Yellow: 0.7543 Black: 0.3853 )"
   ],
   "id": "c3af7153-7445-49b1-b001-a6210dd6889c",
   "name": "PANTONE® PLUS coated"
}
```

8.5.4.20 *Spotcolor Gradations and Rulesets*

8.5.4.2.10 Parameters

8.5.4.2.1.1 GET

Spotcolor gradations exhibit a different format for the "*parameters*" subresource representation than the other resources implementing the generic interface from from 8.2.5.1.

Response JSON Object:
checksum: as in 8.2.5.1.1
updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1

The difference to 8.2.5.1.1 is, that there is no *parameters* attribute in subresource's representation.

Example 1: Gradation

```
  {
      "checksum": "608037728",
      "updated": "Thu, 24 Nov 2016 15:48:37 GMT",
      "validationMessages": [],
      "validationState": "none"
  }
```

Example 2: Rulesets

```
  {
      "checksum": "2112985003",
      "updated": " Wed, 26 Oct 2016 11:28:14 GMT",
      "validationMessages": [],
      "validationState": "none"
  }
```

8.6 ICC Substitutions

8.6.1 Collection

This collection provides a list of all ICC substitution files available on the Server. The collection is located at:

```
colorserver:<port>/resources/substitutions
```

Because there are no subtypes for this resource type. This collection implements the generic subresource collection interface as described in 8.2.3.1.

- •0 GET for listing of ICC substitutions and
- •0 POST for creating new ICC substitution instances.

Substitution files' type extension is *.isr*.

Example:

```
GET /resources/substitutions
```

Response:

```
[
    {
        "description": "",
        "id": "{399144e2-f02f-4ea9-8a40-f39abfd7fbd9}",
        "locked": false,
        "name": "All GMG Conversion Profiles",
        "updated": "Wed, 05 Oct 2016 20:02:32 GMT"
    },
    {
        "description": "Created from legacy ColorServer 4.x hotfolder ... ",
        "id": "{731b369d-741b-4e03-9874-f57e20742832}",
        "locked": false,
        "name": "All GMG Conversion Profiles-migration",
        "updated": "Wed, 26 Oct 2016 11:28:06 GMT"
    }
]
```

8.6.2 Elements

Single ICC substitution rulesets available on the Server are addressed by their ID or name, e.g.:

```
colorserver:<port>/resources/substitutions/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
colorserver:<port>/resources/substitutions/Test%20Substitution%20File
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- •0 GET for retrieving of ICC substitution's metainfo or its resource file contents,
- •0 DELETE for removing of ICC substitution instances,
- •0 PATCH for changing of ICC substitution metainfo attributes and
- •0 PUT for replacing of entire underlying ICC substitution resource files.

8.6.3 Subresources
Every instance contains a single “*parameters*” subresource containing ICC substitution ruleset information. It implements the generic resource “*parameters*” subresource interface, as described in 8.2.5.1, i.e.

- •0 GET for retrieving of ICC substitution processing parameters,

with differences which are described in the following section.

8.6.3.10 *Parameters*

8.6.3.1.10 GET
ICC substitution rulesets exhibit a different format for the “*parameters*” subresource representation than the other resources implementing the generic interface from 8.2.5.1.

Response JSON Object:

checksum: as in 8.2.5.1.1
updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1

The difference to 8.2.5.1.1 is, that there is no *parameters* attribute in subresource‘s representation.

Example:

```
{
    "checksum": "1907929901",
    "updated": " Wed, 05 Oct 2016 20:02:32 GMT",
    "validationMessages": [],
    "validationState": "none"
}
```

8.7 Shared Locations
Shared locations are “packaged” directory paths on a specific machine. They are used in definition of workflows in ColorServer 5 GUI Editor. They are not visible in the workflow representation in REST API though, at least not in the documented part. They can however be used to issue file copy jobs (see 4.1.2.3).

8.7.1 Collection
This collection provides a list of all shared locations available on the Server. The collection is located at:

```
colorserver:<port>/resources/locations
```

Because there are no subtypes for this resource type, this collection implements the generic subresource collection interface as described in 8.2.3.1.

- •0 GET for listing of locations and
- •0 POST for creating new location instances.

Shared location files‘ type extension is *.shloc*.

Example:

```
GET /resources/locations
```

Response:

```
[
   {
       "description": "",
       "id": "{08e6968b-3b16-4438-8ccf-beafd5b78aad}",
       "locked": false,
       "name": "C_Colorserver",
       "updated": "Fri, 04 Nov 2016 10:37:32 GMT"
   },
   {
       "description": "",
       "id": "{858b02c4-28fa-48d2-be7d-a493d0716209}",
       "locked": false,
       "name": "Default",
       "updated": "Wed, 26 Oct 2016 11:16:17 GMT"
   },
]
```

8.7.2 Elements

Single shared locations available on the Server are addressed by their ID or name, e.g.:

```
colorserver:<port>/resources/locations/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
colorserver:<port>/resources/locations/C_Hotfolders
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- GET for retrieving of shared location metainfo or its resource file contents,
- DELETE for removing of shared location instances,
- PATCH for changing of shared location metainfo attributes and
- PUT for replacing of entire underlying shared location resource files.

8.7.3 Subresources

Every instance contains a single *"parameters"* subresource containing shared location's processing parameters. It implements the generic resource *"parameters"* subresource interface, as described in 8.2.5.1, i.e.

- GET for retrieving of shared location's processing parameters.

Example:

```
{
  "checksum": "1067614972",
  "parameters": [
      {
          "id": "Location",
          "value": { "path": "C:\\Colorserver\\" }
      },
      {
          "id": "HostId",
          "value": { "name": "int.gmgcolor.com\\NBDETUSD28" }
      },
      {
          "id": "Enabled",
          "value": { "isEnabled": true }
      }
```

```
    ],
    "updated": "Fri, 04 Nov 2016 10:37:32 GMT",
    "validationMessages": [],
    "validationState": "none"
  }
```

Note: The detailed formats of processing parameters for different resource types is beyond the scope of this specification and are due to change!

8.8 Printers

The *"printers"* collection stores packaged print settings for printers available at a specific computer. Thus if we want to print in two ways on a printer attached to some machine, we will need two instances of printer settings resource in this collection.

Printer settings are used in definition of workflows in ColorServer 5 GUI Editor. They are not visible in the workflow representation in REST API though, at least not in the documented part. They can however be used to issue printing jobs (see 4.1.2.3).

8.8.1 Collection

This collection provides a list of all configured Printer settings available on the Server. The collection is located at:

```
colorserver:<port>/resources/printers
```

Because there are no subtypes for this resource type, this collection implements the generic subresource collection interface as described in 8.2.3.1.

- 0 GET for listing of printer settings and
- 0 POST for creating new printer setting instances.

Printer settings resource file type extension is *.prloc*.

Example:

```
GET /resources/printers
```

Response:

```
[
    {

        "description": "",
        "id": "{7d6757c0-eb88-4f09-82a5-a0a400abba61}",
        "locked": false,
        "name": "HP Office Jet 6950",
        "updated": "Tue, 11 Jul 2017 07:28:59 GMT"
    }
]
```

8.8.2 Elements

Single printer settings available on the Server are addressed by their ID or name, e.g.:

```
colorserver:<port>/resources/printers/ {7d6757c0-eb88-4f09-82a5-a0a400abba61}
colorserver:<port>/resources/ printers/ HP%20Office%20Jet%206950
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- GET for retrieving of printer settings metainfo or its resource file contents,
- DELETE for removing of printer settings instances,
- PATCH for changing of printer settings metainfo attributes and
- PUT for replacing of entire underlying printer settings resource files.

8.8.3 Subresources

Every instance contains a single "*parameters*" subresource containing Printer's processing parameters. It implements the generic resource's "*parameters*" subresource interface, as described in 8.2.5.1, i.e.

- GET for retrieving of Printer's processing parameters.

The parameters contain, among others, Base64-encoded custom print settings and the ID of the machine which the printer is attached to.

Example:

```
{
    "checksum": "3857662171",
    "parameters": [
        {
            "id": "HostId",
            "value": {
                "name": "int.gmgcolor.com\\NBDETUSD28"
            }
        },
        {
            "id": "PrinterData",
            "value": {
                "parameters": [
                    {
                        "id": "PrinterSelection",
                        "value": {
                            "index": 1,
                            "optionId": "HP OfficeJet 6950"
                        }
                    },
                    {
                        "id": "DevMode",
                        "value": {
                            "alignment": 0,
                            "copiesCount": 0,
                            "customParameters":
"SABQACAATwBmAGYAaQBjAGUASgBlAHQAIAA2ADkANQAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAEEAAbcACwIQ78BAgEACQCaCzQIZAABAA8AWAICAAEAWAIDAAEAQQA0AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA ... ",
                            "customParametersSize": "2312",
                            "horizontalOffset": 0,
                            "mirrored": false,
                            "orientation": 0,
                            "pages": 0,
                            "placement": 0,
                            "rescaled": false
                        }
                    }
                ]
            }
        }
    ],
    "updated": "Tue, 11 Jul 2017 07:28:59 GMT",
    "validationMessages": [],
    "validationState": "none"
}
```

Note: The detailed formats of processing parameters for different resource types is beyond the scope of this specification and are due to change!

8.9 Workflows

Workflows are preconfigured and saved action lists to be applied to input files from a given directory. Hotfolders are entities which run workflows by supervising specified input directories and automatically applying workflows' actions to each new file in that directory.

These both terms will be used interchangeably in the following, as their exact boundaries are somewhat blurred.

There are three types of workflows depending on types of their input files: image workflows, PDF workflows, and Job Ticket workflows.

8.9.1 Collection

This collection provides a list of all workflows available on the Server. It is located at:

```
colorserver:<port>/resources/workflows
```

It implements the generic resource collection interface as described in 8.2.2, i.e.

- •0 GET for listing of workflows and
- •0 POST for creating new workflow instances.

8.9.2 Subcollections

8.9.2.10 *Workflow Type Subcollections*

The workflows resource path contains subcollections grouping workflows of a given type. They are correspondingly located at:

```
colorserver:<port>/resources/ workflows/image
colorserver:<port>/resources/workflows/pdf
colorserver:<port>/resources/workflows/jt
```

These subcollections implement generic resource subtype collection interface as described in 8.2.3.1, i.e.

- •0 GET for retrieving the list of subtype collection's elements and
- •0 POST for creating new workflow instances.

The allowed file extensions for different workflow types used in PUT and POST requests are:

Image workflows:	*.imghf*
PDF workflows:	*.pdfhf*
Job Ticket workflows:	*.jthf*

8.9.2.20 *Workflow States Subcollection*

In addition to the standard "*parameters*" subcollection, the "*states*" subcollection enables users to query status of all the workflows present in the system. It is located at:

```
colorserver:<port>/resources/workflows/states
```

8.9.2.2.10 GET

Returns a list of states of all workflows deployed on the Server. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

If-Modified-Since - used to poll for collection's changes (see 2.2.6)

If-None-Match: also used to poll for collection's changes. It does not rely on RFC 1123 date precision as it uses ETags (see 2.2.3)

Response Headers:

Content-Type: *application/json*

Last-Modified: last modification date for the collection

ETag: the entity tag for the collection (i.e. hash of statuses of all hotfolders)

Response JSON Object:

array of JSON objects with following attributes:

hostId: ID of the machine when the workflow is running

name: name of the workflow

id: ID of the workflow

inputFolders: Array of Strings denoting paths of workflow's input directories

outputFolders: Array of Strings denoting paths of workflow's output directories

type: workflow's hotfolder type: *PdfHotfolder*, *ImageHotfolder*, *JtHotfolder*

status: detailed status information for the workflow, see 8.9.4.2.1 for detailed format description

Status Codes:

default (see 2.3.1)

304 Not Modified – If the collection and any of its elements wasn't modified since the time provided in the *If-Modified-Since* header, or if the current entity tag does not differ from the value of *If-None-Match* header.

Example:

```
[
    {
        "hostId": "int.gmgcolor.com\\NBDETUSD28",
        "id": "{1200df75-4dc2-4066-9d4e-066119c48618}",
        "inputFolders": [
            "C:/Hotfolders/PDF/input"
        ],
        "name": "C_Hotfolders test PDF",
        "outputFolders": [
            "C:/Hotfolders/PDF/output"
        ],
        "status": {
            "active": true,
            "consistent": true,
            "files": [],
            "messages": []
        },
        "type": "PdfHotfolder"
    },
    {
        "hostId": "int.gmgcolor.com\\NBDETUSD28",
        "id": "{1200df75-4dc2-4066-9d4e-066119c48618}",
        "inputFolders": [
            "C:/Hotfolders/IMG/input"
        ],
        "name": "C_Hotfolders test IMG",
        "outputFolders": [
            "C:/Hotfolders/IMG/output"
        ],
        "status": {
            "active": true,
            "consistent": true,
            "files": [],
            "messages": []
        },
        "type": "ImageHotfolder"
    }
]
```

8.9.3 Elements

Single workflows present on the Server are adressed in their appropriate subcollections by their IDs or names, or directly from from the top-level *workflows* collection by their ID, according to generic resource collection interface:

```
colorserver:<port>/resources/workflows/pdf/{9e3b5c2a-4640-4a31-b12d-369dd2c32522}
colorserver:<port>/resources/workflows/image/Example%20Image%20Workflow

colorserver:<port>/resources/workflows/{6ee231f0-b406-46ce-85cb-a5a0da5bfe9a}
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- •0 GET for retrieving of workflow metainfo or its resource file contents,
- •0 DELETE for removing of workflow instances,
- •0 PATCH for changing of workflow metainfo attributes and
- •0 PUT for replacing of entire workflow template resource files,

with some extensions which are described in the following section.

8.9.3.10 *PATCH*

In addition to the generic resource instance interface as decribed in 8.2.4.3 this collection defines one more attribute with PATCH semantics:

Request JSON Object: note that per request only one attribute can be set!

name: as in 8.2.4.3
description: as in 8.2.4.3
locked: as in 8.2.4.3
active: run state of the hotfolder, use true to set hotfolder to "*running*" state, false to disable it (*Boolean*)

Example:

```
PATCH /resources/workflows/{2bced842-fcf7-46c4-ba24-b373444e664b} HTTP/1.1
Content-Type: application/json
Host: colorserver:8011

{
    "active": false
}
```

8.9.4 Subresources

Every instance contains a single *"parameters"* subresource containing workflow's processing parameters. It implements the generic resource *"parameters"* subresource interface, as described in 8.2.5.1, i.e.

- •0 GET for retrieving of workflow processing parameters.

with some differences which are described in the following section.

Additionally, the "*status*" subresource with its own interface is defined. It can be used for retrieving of the currnet state of a given workflow.

8.9.4.10 *Parameters*

8.9.4.1.10 GET

Workflows exhibit a different format for the *"parameters"* subresource representation than the other resources implementing the generic interface from 8.2.5.1. If not specified differently, the default type of JSON attributes is *String*.

Response JSON Object:

checksum: as in 8.2.5.1.1
updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1
parameters: as in 8.2.5.1.1
hostId: ID of the machine when the workflow is running
inputFolders: Array of *Strings* denoting paths of workflow's input directories
outputFolders: Array of *Strings* denoting paths of workflow's output directories

Example:

```
GET /resources/workflows/image/{7002a9b4-191c-4b17-829b-2249e4137326}/parameters
```

Response:

```
{
  "checksum": "4018527293",
  "hostId": "int.gmgcolor.com\\NBDETUSD28",
  "inputFolders": [
      " C:/Hotfolders/IMG/input"
  ],
  "outputFolders": [
      " C:/Hotfolders/IMG/output"
  ],
  "parameters": [
      {
```

```
                "id": "Host",
                "value": {
                    "index": 0,
                    "optionId": "int.gmgcolor.com\\NBDETUSD28"
                }
            },
            {
                "id": "JobPriority",
                "value": {
                    "values": { "elements": [ [6] ] }
                }
            },
            ...
        ],
        "updated": "Fri, 25 Nov 2016 07:58:18 GMT",
        "validationMessages": [
            "PDF processing template is invalid or doesn't exist anymore",
            "No processing template was selected"
        ],
        "validationState": "error"
    }
```

Note: The detailed formats of processing parameters for different resource types is beyond the scope of this specification and are due to change!

8.9.4.20 ***Status***

Contains the status of the given workflow. It is located at

```
colorserver:<port>/resources/workflows/:workflow-id/status
```

Please note that this subresource does not allow conditional requests! Monitoring of hotfolder states is instead possible over the "*states*" subcollection 8.9.2.2.

8.9.4.2.10 GET

Returns workflow's status data. If not specified differently, the default type of JSON attributes is *String*.

Parameters:

none

Request Headers:

default (see 2.2.7)

Response Headers:

Content-Type: *application/json*

Response JSON Object:

active: true if the hotfolder with this workflow is running. When workflow is not consistent, this attribute will always be false. Can be set by a PATCH request (see 8.9.3.1).

consistent: true if this workflow's configuration is valid, false otherwise

files: contains an array of hotfolder file status description objects with:

creationTime: time when file was detected in hotfolder (RFC 1123 date)

filePath: full file path in hotfolder's input directory
queueState: processing state of the file - *none, rejected, queued, removed, finished, error, invalid*
rejectionReason: rejection error code - *none, fileTypeErr, nameFilter, fileFormatErr, accesDenied, invalid*
messages: contains an array of file processing message objects with
category: message type - *"none", "info", "warning", "error", "critical"*
text: the actual processing error message
... *:* other, less important attributes

messages: contains an array of validation message objects with:
category: message type - *"none", "info", "warning", "error", "critical"*
text: the actual validation error message
... *:* other, less important attributes

Status Codes:
200 OK – Request completed successfully
404 Not Found – Not found

Example 1: Hotfolder with files waiting to be processed

```
{
    "active": true,
    "consistent": true,
    "files": [
        {
            "creationTime": "25-11-2016 07:54:02.124 GMT",
            "filePath": "C:/Hotfolders/IMG/input/Desert_CMYK_8bit_LZW.tif",
            "messages": [],
            "queueState": "queued",
            "rejectionReason": "none"
        },
        {
            "creationTime": "25-11-2016 07:54:02.130 GMT",
            "filePath": "C:/Hotfolders/IMG/input/FLAG_T24.TIF",
            "messages": [],
            "queueState": "queued",
            "rejectionReason": "none"
        }
    ],
    "messages": []
}
```

Example 2: Hotfolder with invalid configuration

```
{
    "active": true,
    "consistent": false,
    "files": [],
    "messages": [
        {
            "category": "error",
            "iD": 0,
            "source": "Hotfolder Configuration Validator",
            "text": "PDF processing template is invalid or doesn't exist anymore",
            "timestamp": "2016-11-25 08-58-22 290",
            "typeId": "DefaultMessage"
        },
        {
            "category": "error",
            "iD": 0,
            "source": "Hotfolder Configuration Validator",
            "text": "No processing template was selected",
            "timestamp": "2016-11-25 08-58-22 284",
            "typeId": "DefaultMessage"
        }
    ]
}
```

Example 3: Hotfolder with rejected files

```
{
    "active": true,
    "consistent": true,
    "files": [
        {
            "creationTime": "25-11-2016 10:22:57.168 GMT",
            "filePath": "C:/Hotfolders/IMG/input/9353_IndDevN_white.pdf",
            "messages": [
                {
                    "category": "error",
                    "iD": 0,
                    "source": " C_Hotfolders test IMG",
                    "text": "Unsupported file format",
                    "timestamp": "2016-11-25 11-22-57 168",
                    "typeId": "DefaultMessage"
                }
            ],
            "queueState": "rejected",
            "rejectionReason": "fileTypeErr"
        }
    ],
    "messages": []
}
```

8.10 Smart Profiler Resources

Smart profiler resources are resources used exclusively for supporting the *SmartProfiler* product's functionality.

8.10.1 Collection

This collection provides a list of all resources related to smart profiler which are available on the Server. The collection is located at:

```
colorserver:<port>/resources/smartprofiler
```

It implements the generic resource collection interface as described in 8.2.2, i.e.

- •0 GET for listing of smart profiler resources and
- •0 POST for creating new smart profiler resource instances

Example:

```
GET /resources/smartprofiler HTTP/1.1
```

Response:

```
[
    {
        "href": "resources/smartprofiler/documents/{56a4209f-0bd7-4cd8-b70b-
75062aa9452c}",
        "locked": false,
        "name": "Test Smart Profiler 0001",
        "type": "SmartProfiler Wizard Document",
        "updated": "Sat, 15 Jul 2017 12:19:20 GMT"
    },
    {
        "href": "resources/smartprofiler/defaults/db3/{4d303dce-2af0-428c-b285-
422cbca3b087}",
        "locked": false,
        "name": "SpotColors_ISOcoatedv2-39L",
        "type": "SmartProfiler Default",
        "updated": "Sat, 15 Jul 2017 09:45:43 GMT"
    },
    {
        "href": "resources/smartprofiler/defaults/workflows/{076d0392-804b-4785-
9b60-df9ca6369b57}",
        "locked": false,
        "name": "Test Default Workflow 001",
        "type": "SmartProfiler Default",
        "updated": "Sat, 15 Jul 2017 09:46:04 GMT"
    },  ...
]
```

8.10.2 Subcollections

The smart profiler resource path contains two subcollections used correspondingly for smart profiler resources proper (aka. document) and the defaults supporting smart profiler document's creation. They are located at:

```
colorserver:<port>/resources/smartprofiler/documents
colorserver:<port>/resources/smartprofiler/defaults
```

The allowed file extensions for smart profiler resources types used in PUT and POST requests are:

Smart profiler documents:	*.smp*
Smart profiler defaults:	same extensions as for corresponding resource types when used independently

8.10.2.10 *Smart Profiler Document Collection*

This subcollection implements generic resource subtype collection interface as described in 8.2.3.1, i.e.

- •0 GET for retrieving the list of subtype collection's elements and
- •0 POST for creating new spotcolor resource instances.

Example:

```
GET /resources/smartprofiler/documents HTTP/1.1
```

Response:

```
[
    {
        "description": "test test test",
        "id": "{56a4209f-0bd7-4cd8-b70b-75062aa9452c}",
        "locked": false,
        "name": "Test Smart Profiler 0001",
        "updated": "Sat, 15 Jul 2017 12:19:20 GMT"
    }
  ...
]
```

8.10.2.20 *Smart Profiler Defaults Subcollection*

This subcollection implements, as its parent collection did already, again the generic resource collection interface as described in 8.2.2, i.e.

- •0 GET for listing of smart profiler defaults resources and
- •0 POST for creating new smart profiler defaults resource instances.

Example:

```
GET /resources/smartprofiler/defaults HTTP/1.1
```

Response:

```
[
    {
        "href": "resources/smartprofiler/defaults/db3/{4d303dce-2af0-428c-b285-
422cbca3b087}",
        "locked": false,
        "name": "SpotColors_ISOcoatedv2-39L",
        "type": "SmartProfiler Default",
        "updated": "Sat, 15 Jul 2017 09:45:43 GMT"
    },
    {
        "href": "resources/smartprofiler/defaults/workflows/{076d0392-804b-4785-
9b60-df9ca6369b57}",
        "locked": false,
        "name": "Test Default Workflow 001",
        "type": "SmartProfiler Default",
        "updated": "Sat, 15 Jul 2017 09:46:04 GMT"
    },  ...
]
```

8.10.2.30 *Specific Smart Profiler Defaults Subcollections*

The smart profiler defaults resource path contains subcollections corresponding to the smart profiler defaults types, i.e. color templates, normalization templates, PDF temaplates, DB3 databasesand PDF workflows. They are located at:

```
colorserver:<port>/resources/smartprofiler/defaults/color
colorserver:<port>/resources/smartprofiler/defaults/normalization
colorserver:<port>/resources/smartprofiler/defaults/templates
colorserver:<port>/resources/smartprofiler/defaults/db3
colorserver:<port>/resources/smartprofiler/defaults/workflows
```

The allowed file extensions for different smart profiler default resource types used in PUT and POST requests are:

/color	color template files:	see 8.3.2
/normalization	normalization template files:	see 8.3.2
/templates	PDF template files:	see 8.3.2
/db3	DB3 database files:	see 8.5.2
/workflows	PDF workflow files:	see 8.9.2

These subcollections implement generic resource subtype collection interface as described in 8.2.3.1, i.e.

- •0 GET for retrieving the list of subtype collection‘s elements and

- •0 POST for creating new smart profiler defaults resource instances,

Example:

```
GET /resources/smartprofiler/defaults/color HTTP/1.1
```

Response:

```
[
    {
        "description": "",
        "id": "{5340e4b4-c39a-4232-8569-565b127fdb07}",
        "locked": false,
        "name": "InkReport-test-1",
        "updated": "Sat, 15 Jul 2017 09:46:09 GMT"

    }
]
```

8.10.3 Elements

Single smart profiler resources present at the Server are adressed in their appropriate subcollections by their IDs or names.

However, as smart profiler documents and smart profiler defaults are playing totally different role in color processing, the acces over the common root path .i.e. `resources/smartprofiler` is <u>disabled</u>!

```
colorserver:<port>/resources/smartprofiler/documents/{56a4209f...75062aa9452c}
colorserver:<port>/resources/smartprofiler/documents/Test%20Smart%20Profiler%2001

colorserver:<port>/resources/smartprofiler/defaults/{9e3b5c2a...369dd2c32522}
colorserver:<port>/resources/smartprofiler/defaults/db3/{9e3b5c2a...369dd2c32522}
colorserver:<port>/resources/smartprofiler/defaults/db3/SpotColor_ISOcoatedv2-39L
```

The elements implement generic resource instance interface as described in 8.2.4, i.e.

- •0 GET for retrieving of smart profiler resource metainfo or its resource file contents,
- •0 DELETE for removing of smart profiler resource instances,
- •0 PATCH for changing of smart profiler resource metainfo attributes and
- •0 PUT for replacing of entire underlying smart profiler resource resource files,

8.10.4 Subresources

Every instance contains a single "*parameters*" subresource containing smart profiler resource processing parameters. It implements the generic resource instance's "*parameters*" subresource interface, as described in 8.2.5.1, i.e.

- •0 GET for retrieving of smart profiler resource's processing parameters,

with some differences which are described in the following section.

8.10.4.10 *Smart Profiler Defaults*

In case of the smart profiler defaults types, the returned parameters subresource representation differs from the formats specified for the corresponding resource types when queried through their dedicated collections (i.e. *resources/templates/color*, *resources/spotcolor/db3* etc.).

8.10.4.1.1 Parameters

8.10.4.1.1.1 GET

Spotcolor databases exhibit a different format for the *"parameters"* subresource representation than the other resources implementing the generic interface from 8.2.5.1.

Response JSON Object:

checksum: as in 8.2.5.1.1
updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1
resourceTypeId – type of the smart profiler default resource, one of: *ColorNormalization*, *ColorProcessing*, *PdfProcessing* for default templates and *SpotColors*_DB3, *PdfHotfolder* for default DB3 and worklows, respectively (*String*).

The difference to 8.2.5.1.1 is, that there is no *parameters* attribute in subresource's representation.

Example:

```
{
    "checksum": "818256357",
    "resourceTypeId": "ColorProcessing",
    "updated": "Sat, 15 Jul 2017 09:46:09 GMT",
    "validationMessages": [],
    "validationState": "none"
 }
```

8.10.4.2 ***Smart Profiler Documents***

8.10.4.2.1 Parameters

8.10.4.2.1.1 GET

Smart profiler documents exhibit a slightly different format for the *"parameters"* subresource representation than the other resources implementing the generic interface from 8.2.5.1. If not specified differently, the default type of JSON attributes is *String*.

Response JSON Object:

checksum: as in 8.2.5.1.1
updated: as in 8.2.5.1.1
validationMessages: as in 8.2.5.1.1
validationState: as in 8.2.5.1.1
parameters: as in 8.2.5.1.1
printerName – name of the used printer
printerType – type of the used printer
media – name oft he used media
creator – ID of the creator of this smart profiler document

Example:

```
{
"checksum": " 3323852349",
    "parameters": [
            { "id": "PrinterName", "value": "new printer" },
            { "id": "PrinterModeName", "value": "experimental" },
            { "id": "MediaName", "value": "new media" },
            { "id": "SelectedHostId", "value": "int.gmgcolor.com\\NBDETUSD28" },
            { "id": "SelectedSharedLocationId", "value": "" },
            { "id": "SelectedPrinterId", "value": "{7d6757c0-eb88-4f09-82a5-
a0a400abba61}" },
            { "id": "SelectedPrinterPresetId", "value": "" },
            { "id": "IsPrinterCalibrationEnabled", "value": true },
            { "id": "OutputTarget", "value": "printer" },
            { "id": "OutputSubFolderPath", "value": "" },
    ],
    "printerName": "new printer",
    "printerType": "experimental",
    "media": "new media",
    "creator": " Marek.Krajewski",
    "updated": "Wed, 05 Oct 2016 20:03:04 GMT",
    "validationMessages": [],
    "validationState": "none"
  }
```

9 Quick API Overview and Reference

•0 Root

root.getInfo	GET /
root.getSummary	GET /summary

•0 Jobs

jobs.list	GET /jobs
jobs.insert	POST /jobs?job-parameters, POST /jobs { JSON data}
jobs.get	GET /jobs/<job-id>
jobs.getStatus	GET /jobs/<job-id>/status
jobs.setStatus	PATCH /jobs/<job-id>/status { JSON-data }
jobs.getTicket	GET /jobs/<job-id>/ticket
jobs.getResult	GET /jobs/<job-id>/result
jobs.delete	DELETE /jobs/<job-id>
jobs.getClients	GET /jobs/clients
jobs.getLogFile	GET /jobs/logfile
jobs.getInkReport	GET /jobs/inkreport
jobs.printFile	POST /jobs?job-parameters, POST /jobs { JSON data}

•0 Workers

workers.listTypes	GET /workers

•0 Files

files.list	GET /files
files.register	POST /files?file-parameters
files.insert	PUT /files?file-parameters [binary data]
files.getMetadata	GET /files/<file-id> "Accept: application/json"

files.getContents	GET /files/<file-id> "Accept: application/octet-stream"
files.delete	DELETE files/<file-id>

•0 **Licenses**

licenses.list	GET /licenses
licenses.getNames	GET /licenses/types
licenses.isPresent	GET /licenses/<license-name>

•0 **Resources**

resources.list	GET /resources

1. Generic Resource API

<resource>.list	GET /resources/<resource-type>
<resource>.listSubtype	GET /resources/<resource-type>/<subtype>

```
<resource>.insert
   POST /resources/<resource-type>?resource-parameters [ binary data ]
   POST /resources/<resource-type>/<subtype>?resource-parameters [ binary data ]

<resource>.replace
   PUT /resources/<resource-type>/<resource-id> [ binary data ]
   PUT /resources/<resource-type>/<subtype>/<resource-id> [ binary data ]
   PUT /resources/<resource-type>/<subtype>/<resource-name> [ binary data ]

<resource>.getMetadata
   GET  /resources/<resource-type>/<resource-id> "Accept: application/json"
   GET /resources/<resource-type>/<subtype>/<resource-id> "Accept: application/json"
   GET /resources/<resource-type>/<subtype>/<resource-name>
       "Accept: application/json"

<resource>.getFileContents
   GET  /resources/<resource-type>/<resource-id> "Accept: application/octet-stream"
   GET /resources/<resource-type>/<subtype>/<resource-id>
       "Accept: application/octet-stream"
   GET /resources/<resource-type>/<subtype>/<resource-name>
       "Accept: application/octet-stream"

<resource>.getParameters
   GET /resources/<resource-type>/<resource-id>/parameters
   GET /resources/<resource-type>/<subtype>/<resource-id>/parameters
   GET /resources/<resource-type>/<subtype>/<resource-name>/parameters

 <resource>.setMetadata
   PATCH /resources/<resource-type>/<resource-id> { JSON data }
   PATCH /resources/<resource-type>/<subtype>/<resource-id> { JSON data }
   PATCH /resources/<resource-type>/<subtype>/<resource-name> { JSON data }
```

```
<resource>.delete
    DELETE /resources/<resource-type>/<resource-id>
    DELETE /resources/<resource-type>/<subtype>/<resource-id>
    DELETE /resources/<resource-type>/<subtype>/<resource-name>
```

2. Additional Resource APIs

```
templates.getTypes        GET /resources/templates/types

db3.listColorSets         GET /resources/spotcolor/db3/<db3-id>/sets
db3.getColorSet           GET /resources/spotcolor/db3/<db3-id>/sets/<set-name>

workflows.listStates        GET /resources/workflows/states

workflows.getStatus
     GET /resources/workflows/<workflow-id>/status
     GET /resources/workflows/<workfl-subtype>/<workflow-id>/status
     GET /resources/workflows/<workfl-subtype>/<workflow-name>/status
```