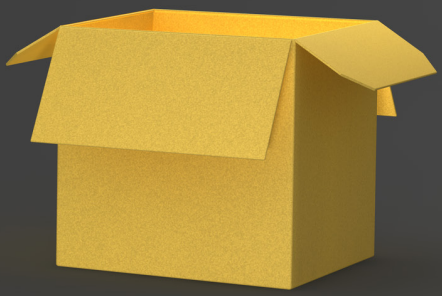
FSI Server

The Complete Single Source Imaging Solution

OpenAPI Documentation



OpenAPI Documentation for FSI Server

Developed by:

NeptuneLabs GmbH
Lagesche Str. 32
D-32657 Lemgo
Germany

Last updated: September 2013
FSI Server Version 3.0
Manual Revision 004

Table of Contents

OpenAPI Documentation
for FSI Server 3.0

1. Introduction

This document provides an insight in how to use the FSI Server from a developers point of view. Whereas the manual describes the setup, administration and every day usage of the FSI Server, this guide is directed to developers integrating FSI Server in complex environments or designing client applications. The FSI Server offers an extensive API providing access to the full functionality of the server.

This document refers to FSI Server 3.0 or above, although most parts also apply to previous versions. For maintainers of thrid party applications using the FSI Server OpenAPI will find the changes to previous versions explicitly listed in Appendix B.

The API can be considered to be split into two main parts. The first part is the servers core task: image and meta data delivery in real time. FSI Server is highly configurable to deliver the content needed in the correct format needed. The second part of the API is a REST-like interface to manage the data on the server. It allows basic operations like uploading and deleting items from the server. All parts of the API use the standardized Hypertext Transfer Protocol (HTTP) allowing applications to be devloped in every modern programming language.

2. Retrieving Data from the Server

The FSI Servers primary task is to provide quick access to images, image ranges and image meta data, mainly for web usage. The available output formats for images are therefore limited to what modern browsers can display. The output format for meta data can be modified to match the developers needs by providing templates.

2.1 Request Types

All HTTP requests to this part of the server API are GET or POST requests. Every request must contain a mandatory parameter named type which specifies the expected return value. Currently available types are *image*, *info*, *list*, *search* and *appinfo*. The *appinfo* type is only meant for usage by NeptuneLabs Client applications like FSI Viewer. The other types are discussed in detail in the following sections. Image, info and list requests are always directed at a certain renderer which defines the internal methods used to handle the request. Renderers are defined by XML files in a directory called *renderers* in the FSI Servers main config directory. More detailed information on renderers can be found in the FSI Server manual in Part 1, chapter 3.5.8.

2.2 Image Requests

In general, image requests are GET requests and are identified by the value *image* for the parameter type. Except for error cases, responses to image requests will always contain image data. The resulting image format depends on the renderer addressed. The renderer also defines the specific image-renderer used and the optionally applied image effects.
The image request is evaluated and depending on the renderer settings, the request is passed on to the image-renderer which assembles the image from the data stored in the internal storage or the multiresolution files. If any effects are specified in the renderer or in the request parameters they will then be applied before the image is finally converted and compressed to the format defined in the renderer.
A typical image request therefore has the following parameters:

Parameter	Description	Default
Type	Mandatory parameter specifying the request type	–
Renderer	Specifies the renderer to be used.	Defined in server configuration
Source	Mandatory, specifies the image requested	–
Effects	Optional effects to apply to the resulting image	–
Quality	Optional parameter used to define the compression quality when requesting a jpg image	Defined in the renderer

Depending on the image-renderer used there can be more mandatory or optional parameters.

2.2.1 Default Image Renderer

The default image-renderer is used if no other image-renderer is specified in the addressed renderer. The default image-renderer is the first choice in most scenarios like (3D-)product presentation or page rendering for FSI Pages. It allows requesting complete images as well as image ranges, scaling the images to the requested size. Depending on the configuration it will keep the source images aspect ratio or distort the image to exactly fit the requested size. The default image-renderer is configured by setting the following configuration options in the imagerenderer section of the renderer definition.

Option	Description	Default
keepAspectRatio	Defines if the original aspect ratio is preserved or if the image(-region) will be distorted to exactly match match the requested size	true
maxWidth, maxHeight	Defines the Maximum size of images that can be requested through this renderer	Height: 3000 pixels Width: 3000 pixels
defaultWidth, defaultHeight	Default image dimensions used when none are given in the URL	Height: 300 pixels Width: 300 pixels

The default image-renderer allows using the following additional parameters in requests:

Parameter	Description	Default
width & height	Both optional, these parameters specify the size of the image requested.	Defined in the profile.
left, top, right, bottom	Optional parameters with values ranging from 0 to 1. Used to request an image region.	left=0, top=0, right=1, bottom=1
rect	Four comma-separated values ranging from 0 to 1 which can be used instead of left, to, right and bottom	0,0,1,1

2.2.2 Double Page Image Renderer

The double page image-renderer composes two images to one image by placing them next to each other. As the name suggests, this is primarily designed for displaying two adjacent pages of a catalog or brochure. The first client side FSI application to make use of this is FSI Pages mobile which uses this image-renderer to display the catalog overviews. The Double Page Image Renderer does currently not support image ranges, so using this renderer to zoom into images using FSI Viewer is not possible.

The image-renderer configuration section in the renderer definition allows setting the following parameters:

Option	Description	Default
maxWidth, maxHeight	Defines the maximum size of images that can be requested through this renderer	3000 x 3000
defaultWidth, defaultHeight	Defines the default dimensions to use, if none are provided in the request.	300 x 300
backgroundColor	Defines the background color, which is visible as soon as one of the images has transparent regions or if the two images have different aspect ratios.	FFFFFF
defaultAlignment	Defines how to align image on a page, if they do not fill out the complete page.	CC

This image-renderer supports a number of query parameters, some mandatory, others optional.

Query Parameter	Description	Type	Default
source	Needs to provide two comma separated image urls. The first one for the left and the second for the right page.	mandatory	–
width & height	The requested image dimensions.	optional	see defaultWidth and defaultHeight options above
leftalignment, rightalignment	Defines how to align the image on the left/right page.	optional	see default Alignment above
leftimagecrop, rightimagecrop	Defines a range to which the image is cropped before placing it on the left/right page. (four comma-separated floating point values specifying the top-left corner and the width and height)	optional	0, 0, 1, 1
leftinnergap, rightinnergap	Defines an inner gap between the center of the resulting double-page and the image edges (floating point value between 0 and 1 specifying the gap size in relation to the image size).	optional	0

2.2.3 Anaglyph Image Renderer

The Anaglyph Image-Renderer provides the basic functionality for using FSI Server to generate stereoscopic 3D images. It composes an anaglyph image

from two given source images, allowing to specify the type of 3D glasses used to view the image by providing the color filters for the left and right image. An additional parameter allows applying effects to the image prior to performing the necessary color and compose operations.

The image-renderer supports the following configuration parameters in the renderer declaration:

Option	Description	Default
maxWidth, maxHeight	Defines the max. size of images that can be requested through this renderer.	3000 x 3000
defaultWidth, defaultHeight	Defines the default diemsions to use, if none are provided in the request.	300 x 300

The query parameters allow flexible use cases whilst only implying a minimum of mandatory parameters. The basic parameters are the same as those used in the default image-renderer.

Query Parameter	Description	Type	Default
source	Defines the two source images from which the resulting image is composed.	mandatory	–
width & height	Width and height of the resulting image.	optional	see default values above
top, left, bottom, right	Parameters with values ranging from 0 to 1. Used to request an image region.	optional	top: 0, left: 0, bottom: 1, right: 1
rect	Four comma-separated values ranging from 0 to 1 which can be used instead of left, top, right and bottom	optional	0, 0, 1, 1
colormode	Two comma separated colors defining the color filter for the left and right image	optional	red,cyan
preeffects	Allows specifying image effects that will be applied to the source images before composition. For details on the effect syntax, see Part II section 7 in the FSI Server manual.	optional	–

2.3 Template System

Info and List requests are also handled by specific renderers. This allows delivering meta data that matches the images produced by the different image renderers. All info- and list-renderers use a template system to actually render the gathered meta data. By supplying custom templates the format of the resulting responses can therefore be easily adapted to the needs of a client application. The template system used is Freemarker. A full documentation of the freemarker language can be found on the website at http://www.freemarker.org. The meta data provided in the templates depends on the renderer and can be found in the tables in the following sections. The template the renderer will use can be specified using the *template* parameter in the request. Each template is made up of an attributes section and the actual rendering information. The attributes section is evaluated at the beginning of the rendering process and can therefore contain information the renderer needs prior to the actual rendering. The supported attributes are listed in the following table:

Attribute name	Description	Example / Possible Values
datasources	Defines a list of datasources used in this template. For performance reasons only the image meta data specified here will be available when rendering the template	'IPTC':'imagepool' 'EXIF':'imagepool' 'XMP':'imagepool' or a comma-separated list containing any number of the above
headers	Defines response headers returned when processing HTTP requests	'Content-Type':'application/json; charset=UTF-8'

2.4 Info Requests

Info requests are designed to deliver the meta data matching an image request. An info request in general can provide access to all information available on a single or multiple images, although specific renderers might only provide basic information.

A typical info request has at least the following parameters:

Parameter	Description	Example / Possible Values
type	Mandatory parameter specifying the request type	–
renderer	Specifies the renderer to be used.	Defined in server configuration
source	Mandatory, specifies the image requested	–
template	The template used to render the response	Defined in the server configuration

The current version of FSI Server comes with one Info renderer only.

2.4.1 Default Info Renderer

The default info-renderer provides access to all meta data available for a single image. This includes file specific data like the last modified date and the file size, as well as image specific data like the contents of IPTC and EXIF fields.

The following table includes all variables possibly available in the templates.

Parameter	Description
info.src	The file source path
info.size	The file size
info.width	The image with
info.height	The image height
info.lastmodified	The files last modification time
info.alpha	True if the image contained an alpha channel
info.importstatus	The current import status: 0 - unknown 1 - imported 2 - queued 3 - error
info.iptc	An array containing the IPTC information
info.exif	An array containing the EXIF information

The templates for info requests delivered with FSI Server are located in WEB-INF/internal/templates/info in the installation directory. These should not be modified as they are used by the webinterface and the FSI Viewer with all its add-ons. Custom templates can simply be placed in the folder WEB-INF/templates/info. Their filename must end with the extension ".ftl".

As described in the previous chapter the *define* attribute in the templates allows specifying template specific parameters used for rendering. The default info renderer currently acknowledges only one value:

Define attribute	Description	Default
escape	Defines the method used to escape special characters. Possible values are: NONE – No escaping. XML – Escapes the XML markup delimiters as well as single and double quotation marks. JSON – Escapes the data for usage in JSON objects.	NONE

The following example shows one of the templates delivered with FSI Server and has comments on all important statements describing their function. Using this template results in an XML response containing basic file data and selected iptc meta data.

```
<#ftl encoding="UTF-8" attributes={
'define':{
     'Escape':'XML'
}
}>
<?xml version="1.0" encoding="UTF-8" ?>
<fsi:FSI xmlns:fsi="http://www.fsi-
                    viewer.com/schema">

 <Image>
   <Path value="${info.src}"/>
   <#if info.width??><Width
        value="${info.width}"/></#if>
   <#if info.height??><Height
        value="${info.height}"/></#if>
 </Image>

 <#if info.iptc??>


   <Options>
    <#if info.iptc["FSI SceneSets"]??>
     <SceneSets>
      ${info.iptc["FSI SceneSets"]}
     </SceneSets>
    </#if>
    <#if info.iptc["Caption"]??>
     <iptc_caption>
      ${info.iptc["Caption"]}
     </iptc_caption>
    </#if>
   </Options>
  </#if>
</fsi:FSI>
```

Template attribute definitions

Static code which comes out unmodified.

Prints the image file path and the width and height of the image

Checks if the is any IPTC data available

Prints the contents of the FSI SceneSets and the Caption IPTC Fields.

2.5 List Requests

List-renderers deliver lists of images and directories. They can be used to develop client application that allow browsing through the directory structure as well as applications that present a set of images like FSI Pages and FSI Showcase. As with Info requests the output format can be adapted by providing custom templates. The current version of FSI Server comes with one list renderer only.

2.5.1 Default List Renderer

The default list-renderer allows accessing directory meta data as well as basic image meta data like image dimensions, file size and import status. Detailed image meta data like the contents of IPTC and EXIF fields is not available for performance reasons. The full list of variables available in the list templates when using this renderer is shown in the following table

Parameter	Description
currentDir	The path to the current directory
summary. entryCount	The total number of entries in the list
summary. imagecount	The number of images in the list
summary. directoryCount	The number of directories
summary. completeCount	
summary. lastModified	The last modified date of the requested directory
restrictions. readOnly	True if the directory is read-only
restrictions. writeEnabled	
restrictions. downloadOrigin	True if downloading the original image files from this directory is permitted
ilist	A list containing entries for the images and subdirectories in the requested directory. Each entry has a property type which is either "directory" or "image". Depending on the type, the entries have more properties: Image entries have the info properties described in XX (Default Info Renderer), whereas directory entries have two properties: hasSub, which is true if the directory has further subdirectories and sub which contains the number of subdirectories.

2.6 Search Requests

Search requests result in lists of images matching certain search queries. The output is rendered using the list templates, so the output format can be modified by providing custom templates and placing them in the WEB-INF/templates/list folder. The search syntax is described in detail in section 3.5. in the FSI Server manual.

3. Modifying Data

As mentioned in the introduction, the part of the API used to manage images on the server is also HTTP based and uses a REST-like style. Strictly speaking it is not REST as it requires cookies for authentication and provides upload functionalities via POST in addition to the PUT method to allow writing parts of the client applications in flash.

3.1 Interface Overview

The API allows creating, deleting and renaming directories as well as uploading, deleting, renaming and downloading files. Actions performed on directories are always addressed using URLs of the form:

http://your.fsi-server.com/fsi/service/directory/path/to/directory

Actions perfomed on files are always addressed using the following syntax:

http://your.fsi-server.com/fsi/service/file/path/to/file

In addition to the two URLs above, login and logout requests are necessary to create and destroy the session and keep alive requests can be sent to prevent sessions from expiring. These requests need to be addressed at:

http://your.fsi-server.com/fsi/service/login
http://your.fsi-server.com/fsi/service/logout
and
http://your.fsi-server.com/fsi/service/sessionrefresh

All responses to requests will be XML or JSON depending on the accept header sent in the request. Commonly used responses are listed in section 3.7, including examples of what they will look like in XML and JSON format as well a detailed description of the response content.

3.2 Authentication

The authentication is handled by FSI Server itself to be independent of the application server and allow a more flexible user management.
In order to authenticate itself a client application must be able to handle cookies. If it has received a cookie from the server, this cookie must be sent with every following request to the server.
Two requests are necessary to authenticate the client application. The first is a GET request addressed at the Login URL:

http://your.fsi-server.com/fsi/service/login

This returns a JSON or an XML response. If the server is not ready to authenticate users, the response will contain an error message describing the problem. If the server is ready the response will describe how the password needs to be submitted

to the FSI Server as a value for *loginmethod*. By default the value will be *hash*, stating that a password hash is submitted instead of the password itself. If FSI Server is configured to autheticate against a Kerberos Server though, then the password itself is required by the server. If the login method is stated to be *hash*, then the response also contains a salt. The exact format of the response and the possible values is described in 3.7.2 (SaltResponse). The client application must then use the salt to create a password hash using a SHA-256 algorithm of the form

hash = sha256(salt + sha256(password))

where the plus is the concatenation of two strings. The resulting hash or the plain password is then posted to the Login URL as value of a variable called *password* together with the login name as value of *username*. The response to this post request will include a state which is either success of failed and a message including details on the failure. In case of success the response will also contain the number of seconds until the valid session will expire. This response is described in 3.7.3 (LoginResponse).
If an inactive client application wants to extend the session without performing any actions it can send GET requests to

http://your.fsi-server.com/fsi/service/sessionrefresh

If the session is valid the response will contain the number of seconds until the session will expire. See also 3.7.4 (SessionRefreshResponse)
When the user logs out or when the client application has completed its tasks or exits it should log out. This is done by sending a GET request to

http://your.fsi-server.com/fsi/service/logout

This destroys the session and returns an empty response.

3.3 Managing directories

Provided the logged in user has the appropriate permissions, directories can be created, renamed or deleted using the API. All requests need to be directed at the directories URI:

http://your.fsi-server.com/fsi/service/directory/path/to/directory

All requests will be responded to with XML or JSON responses, depending on the accept-header sent.

3.3.1 Creating directories

To create a directory on the server the client application has to send a PUT request to the directory URI. This requires that at least the source connector exists (in the example it would have to be called "path"). The response is an ActionResponse (see Section 3.7.1) and will inform about success or failure.

3.3.2 Deleting directories

Directories can the deleted by sending an HTTP DELETE request to the directory URI. The response to a delete request will also be an ActionResponse. Sending a delete request will recursively delete the directory, including all subdirectories and images. Deleted directories and images are deleted from the servers internal storage and the source folder. FSI Server does not keep backups, so deleted images can not be restored.

3.3.3 Renaming directories

Directories can be renamed by sending a POST request to the directory URI. The POST request must contain a form parameter called to with the new pathname of the directory as value. As renaming will affect all subdirectories and included images, renaming directory containing lots of data will result in massive restructuring of the internal storage. Renaming large directories therefore might take a while. The POST request to rename directories will return an ActionResponse.

3.3.4 Re-importing directories

Although it should not be necessary a client application can trigger the re-import of all images in a directory. To schedule a directory for re-import a POST request containing a form parameter named *reimport* with the value *true* needs to be sent to the directory URI. Depending on the current number of images in the import queue the re-import might not take place immediately. As long as the re-import has not started the data from the original import is accessible from the FSI Server. As most requests to modify data on the server, this too will return an ActionResponse.

3.4 Managing Files

The operations provided to manage files are similar to those provided for directories. The API allows uploading, deleting, renaming and downloading files. Manipulation of images directly on the server is not supported. FSI Server 3 allows manipulating file metadata, as long as the file is used in FSI Servers context. The manipulated metadata is not written to the source file. All operation requests need to be addressed to the File URI:

http://your.fsi-server.com/fsi/service/file/path/to/file

3.4.1 Uploading Files

Files can be uploaded by sending an HTTP PUT request to the file URI. The body of the request should contain the file data or if the last modified date of the image is relevant, an XML or JSON document wrapping the file. In the latter case the XML root node should be called "file" and it should contain

the nodes "lastModified" (specifiying the date as a Unix timestamp) and "data" containing the file data. A JSON representation should contain the two values as object fields. To allow checking if a file can be uploaded prior to transmitting the file data, the client application can send a POST request including the desired location and filesize. This request must be addressed to

http://your.fsi-server.com/fsi/service/postupload

The content type should be "application/x-www-form-urlencoded" and the request should include the form parameters *filesize*, *filename*, *dir* and *lastmodified*. The response will be an ActionResponse and will inform the Client about any problems that might occur if an upload is attempted. Not all file type can be uploaded to arbitrary locations. For storage and multiresolution source connectors, only known image files are accepted.
As the Adobe Flash browser plug-in allows selecting multiple files but cannot send PUT requests the API also allows uploading files through POST requests. The POST request must be directed at the file URI and the response to this request is also an ActionResponse. The possible return values and status codes for the upload and pre-upload requests can be found in the following table.

Code	Description
2xx	Ok resp. proceed with upload
310	File too large
311	Unusable data uploaded
401	Not authorized (Session expired)
403	Forbidden (Access to directory denied)
404	Target directory does not exist
409	Invalid Image Path
413	Insufficient storage space

When uploading large files it can be helpful to keep track of the upload progress. In order to allow this, the FSI Server API allows requesting the current state of the upload by sending a GET request to:

http://your.fsi-server.com/fsi/service/uploadstatus/path/to/file

The response to this request will be an XML or JSON containing the number of bytes already uploaded as well as the total number of bytes expected.

3.4.2 Downloading files

If permitted by the source connector, the original files can be downloaded. A GET request directed at the file URI will result in a download of the unmodified source file if the request oes not contain an accept header preventing an

image/* type. If the requests only accepts application/xml or application/json, then the response will contain an image message body containing metadata as described in 3.7.9.

3.4.3 Deleting files

Just like directories, files can be deleted by sending an HTTP DELETE request to the file URI. If permitted, the file will be deleted from the source connector directory as well as from the internal storage if appropriate. By default the deleted files will be moved to the trash folder and are not deleted permanently.

3.4.4 Renaming / moving files

The renaming resp. moving is performed through an POST request directed at the file URI. The request body needs to contain a parameter named to with the new path/filename combination as value. Again, the response to expect is an ActionReponse. When moving files between different source connectors, the move request might be rejected if the target source connector does not support the specified file type.

3.4.5 Re-importing Images

A re-import of an image can be triggered by sending a POST request to the file URI with a parameter named reimport and a value of true. As documented in section 3.3.4 the re-import must not necessarily take place immediately but is instead appended to the end of the source managers import queue.

3.5 Managing trash

FSI Server 3 places all deleted files in a trash directory located within the internal storages directory structure. A user/client application can list the contents of the trash he has access to, by sending a standard list request with *_trash* as a value for the *source* parameter.
The response will contain an ID to reference the trash entry as well as a description including the original files location. this ID can the be used to as a source for a move request as described in 3.4.4.

3.6 Download and Batch Processing

In addition to downloading single files as described in section 3.4.2, FSI Server 3 adds a feature to compose an archive of multiple source files as well as multiple rendered images. All operations regarding the control of batch processes are POST requests directed at:

http://your.fsi-server.com/fsi/service/jobqueue

The mandatory and optional form parameters are summarzied in the following table:

Parameter	Description	Type	Default
cmd	the command to execute createAndStart cancel restart	mandatory	-
name	the name of the new archive. Applies *createAndStart* only	-	-
archiveType	the type of the new archive (zip, tar.gz or tar.bz2). Applies *createAndStart* only	-	zip
file	path and filename of the file to add to the archive. Multiple occurences supported. Applies *createAndStart* only	mandatory for *createAndStart*	-
renderingQuery	if present the rendering option will be applied to all images before archiving. Applies *createAndStart* only	optional	-
scheduleDate	a Unix timestamp when to start processing. Applies *createAndStart* only	optional	-
id	the id of the job to restart or to cancel	mandatory for *cancel* and *restart*	-

Once submitted a queued job and its id (for canceling and restarting) can be found in the list of batch-jobs by sending a standard list request using *_download* as value for the *source* parameter. The list will also containg the current status and, for running processes, the current progress.
Archives created by completed jobs can be downloaded using a standard download request as described in section 3.4.2.

3.7 Preferences

FSI Server allows setting global and user specific preferences for client applications to use. These are simple key-value pairs and can be set by editing XML files on the server or using the administration interface provided by FSI Administrator. More on setting preferences can be found in the FSI Server manual. A client application can access the preferences by sending a GET request to the following URL

http://your.fsi-server.com/fsi/service/prefs

The result will be an XML or JSON representation of the preferences stored on the server for the currently logged in user. Modifying the preferences by the client application itself is not supported.

3.8 Changing the password

Changing the password requires more than a single request. For security reasons the new password should not be transmitted in plain text and the old password is required to allow client applications to include measures against captured sessions. The password change consists of two requests, both directed at the password resource URI: http://your.fsi-server.com/fsi/service/password. The first request is a GET request to acquire a salt. Similar to the login procedure, this salt is used to create a hash of the old password. This hash is then transferred together with an SHA-256 hash of the new password wrapped in an XML message. This message needs to be sent to the password URI using a put request. The exact format of the required message can be seen in 3.8.11.

3.9 Messages used in Management Requests

On the serverside the request and response bodies are represented by Java objects derived from classes containing JAXB annotations. This chapter discusses the message types in detail and provides the XML Schema documents describing the message content. Depending on the accept header sent by the client the server will return either the XML or JSON representation of a message. The XML format can be derived directly from the XSD provided. The JSON messages are created by Jettison using the "mapped" notation.
If the client application is implemented using Java, developers do not need to spend time on parsing message bodies. Instead the classes contained in the FSI Server Development Pack can be used together with the JAX-RS Client API to easily develop clients without the needs to manually implement the communications part. An example Java codesnip can be found in Section 3.9 and is also part of the development pack. If the XML Schema definitions are needed to derive implementations in other languages they can easily be created from the Java files in the FSI Server Dev Pack using schemagen which is part of the current Java SE 6 versions.

3.9.1 Action Response

The Action Response is used to confirm a request has completed successfully or to notify the client application of any errors that prevented the FSI Server from completing the action.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
               "http://www.w3.org/2001/XMLSchema">
 <xs:element name="response"
             type="actionResponse"/>
 <xs:complexType name="actionResponse">
  <xs:sequence>
    <xs:element name="cause" type="xs:string"
                minOccurs="0"/>
    <xs:element name="details" type="xs:string"
                minOccurs="0"/>
    <xs:element name="statuscode" type="xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Action Response Message

Depending on the request headers the Action Response can be formatted as XML or JSON. The following shows two example return messages.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
 <statuscode>311</statuscode>
 <cause>Unsupported filetype or broken image
</cause>
</response>
```

Example Action Response in XML format

```
{
 "statuscode":"311",
 "cause":"Unsupported filetype or broken image"
}
```

Example Action Response in JSON format

3.9.2 SaltResponse

The SaltResponse is returned when addressing a GET request at the Login URL. It contains the login method stating how the send the password and optionally a salt, or a message why the server is not ready for users or client applications to log in.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"xmlns:xs=
               "http://www.w3.org/2001/XMLSchema">
 <xs:element name="saltResponse"
             type="saltResponse"/>
 <xs:complexType name="saltResponse">
  <xs:sequence>
   <xs:element name="state" type="xs:string"
               minOccurs="0"/>
   <xs:element name="salt" type="xs:string"
               minOccurs="0"/>
   <xs:element name="message" type="xs:string"
               minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Salt Response Message

3.9.3 LoginResponse

The Login Response will be sent as reply to a POST request directed at the login URL. If the transmitted credentials were verified and the login succeeded the response will contain a success state and the number of seconds before the session will expire, if no further requests are received. If it fails a message describing the reason for the failure will be included.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
               "http://www.w3.org/2001/XMLSchema">
 <xs:element name="loginresponse"
             type="loginResponse"/>
 <xs:complexType name="loginResponse">
  <xs:sequence>
   <xs:element name="state" type="xs:string"
               minOccurs="0"/>
   <xs:element name="messageCode" type="xs:int"/>
   <xs:element name="message" type="xs:string"
               minOccurs="0"/>
   <xs:element name="expiry" type="xs:long"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Login Response Message

3.9.4 SessionRefreshResponse

The SessionRefreshResponse is returned when addressing requests to the session refresh URL described in 3.2. The response is made up of a single value containing the number of seconds until the session expires.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
                "http://www.w3.org/2001/XMLSchema">
 <xs:element name="sessionrefreshresponse"
             type="sessionRefreshResponse"/>
 <xs:complexType name="sessionRefreshResponse">
  <xs:sequence>
   <xs:element name="expiry" type="xs:long"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Session Refresh Response Message

3.9.5 PrefsResponse

The PrefsResponse is returned when requesting the users preferences (see 3.5). It contains a list of keys assigned to arrays of values.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
                "http://www.w3.org/2001/XMLSchema">
 <xs:element name="prefs" type="prefsResponse"/>
 <xs:complexType name="prefsResponse">
  <xs:sequence>
   <xs:element name="pref" type="pref"
               minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="pref">
  <xs:sequence>
   <xs:element name="value" type="xs:string"
               minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"/>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Prefs Response Message

3.9.6 UploadStatusResponse

The UploadStatusResponse is sent when requesting information on the progress of currently running uploads as described in 3.4.1.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
                "http://www.w3.org/2001/XMLSchema">
 <xs:element name="UploadStatusResponse"
             type="uploadStatusResponse"/>
 <xs:complexType name="uploadStatusResponse">
  <xs:sequence>
   <xs:element name="uploaded" type="xs:long"/>
   <xs:element name="total" type="xs:long"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Upload Status Response Message

3.9.7 BooleanResponse Message

This message type is only used to respond to logout requests. It will contain a value of either true or false.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
                "http://www.w3.org/2001/XMLSchema">
 <xs:element name="booleanResponse"
             type="booleanResponse"/>
 <xs:complexType name="booleanResponse">
  <xs:sequence>
   <xs:element name="value" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Boolean Response Message

3.9.8 Image Message

This message type is used when sending PUT requests to file URIs in order to upload files as described in 3.4.1 or as response message body when sending GET requests with the appropriate accept header to file URIs. When using this message type in conjunction with GET requests the data part will be left empty, so that this kind of request can be considered as a simplified info request. This is only implemented for completeness. Large scale client applications should consider using info requests as described in 2.4 instead.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
               "http://www.w3.org/2001/XMLSchema">
 <xs:element name="image" type="image"/>
 <xs:complexType name="image">
  <xs:sequence>
   <xs:element name="data" type="xs:base64Binary"
               minOccurs="0"/>
   <xs:element name="fileSize" type="xs:long"/>
   <xs:element name="lastModified"
               type="xs:long"/>
   <xs:element name="targetPath" type="xs:string"
               minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Image Message

3.9.9 Directory Message

Used in the message body to respond to GET requests directed at directory URIs. These requests are comparable to the list requests described in section 2.5. but provide only file system based information.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs=
               "http://www.w3.org/2001/XMLSchema">
 <xs:element name="dirEntry" type="dirEntry"/>
 <xs:element name="directory" type="directory"/>
 <xs:element name="imageEntry" type="imageEntry"/>
 <xs:complexType name="imageEntry">
  <xs:sequence/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="lastModified" type="xs:long"
                use="required"/>
  <xs:attribute name="size" type="xs:long"
                use="required"/>
 </xs:complexType>
 <xs:complexType name="dirEntry">
  <xs:sequence/>
  <xs:attribute name="name" type="xs:string"/>
 </xs:complexType>
 <xs:complexType name="directory">
  <xs:sequence>
   <xs:element name="image" type="imageEntry"
               minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="subdirectory" type="dirEntry"
               minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the Directory Message

3.9.10 Password Message

This message format is used as a message body in the PUT requests sent to the server in order to change the password. For details on how the complete procedure of a password change looks like, please refer to chapter 3.7.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="passwordchange" type="password"/>
 <xs:complexType name="password">
  <xs:sequence>
   <xs:element name="newpasswordhash"
               type="xs:string" minOccurs="0"/>
   <xs:element name="oldpasswordhash"
               type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

XML Schema for the password message.

3.10 Example Code

The following Java codesnip shows how easy it is to develop a client application using FSI Servers API. Only a few lines of code are necessary to log in, upload an image and log out again. The example requires the FSI Server Dev Pack which can be downloaded from http://www.fsi-viewer.com as well as the Jersey Client bundle available at http://jersey.dev.java.net. The FSI Server Dev Pack contains a jar file which includes the JAXB classes representing the request and response messages as described in 3.6.

```
// initialize client
ApacheHttpClientConfig clientconfig =
       new DefaultApacheHttpClientConfig();
clientconfig.getProperties().put(
       ApacheHttpClientConfig.PROPERTY_HANDLE_COOKIES,true);
Client c = ApacheHttpClient.create(clientconfig);

// send a GET request to obtain the salt
WebResource r =
 c.resource("http://your.fsi-server.com/fsi/service/login");
SaltResponse sr = r.get(SaltResponse.class);
String salt = sr.salt;

// combine password with salt and generate login hash
String passwordhash = randomHelper.sha256(password);
String loginhash = randomHelper.sha256(salt + passwordhash);

// send login request
Form form = new Form();
form.add("username", username);
form.add("password", loginhash);
LoginResponse lr = r.post(LoginResponse.class, form);
if (lr.state.equals("success")) {

  // create object to upload
  Image i = new Image();
  File imagefile = new File(testfile);
  byte[] filedata = ... // read from disk …
  i.setData(filedata);
  i.setFileSize(imagefile.length());
  i.setLastModified(imagefile.lastModified());

  // create resource to upload the image to
  WebResource imageresource
   = c.resource("http://your.fsi-server.com/"
     + "fsi/service/image/samples/image.tif");

  // upload the image using a PUT request
  ActionResponse ar =
```

```
        imageresource.put(ActionResponse.class, i);

    [ … check status and handle errors … ]

    // logout
    WebResource logoutresource
     = c.resource("http://your.fsi-server.com"
       + "/fsi/service/logout");
    logoutresource.get(BooleanResponse.class);
  }
```

Appendix

A. Links to resources

http://www.fsi-viewer.com: Home of the FSI Viewer and FSI Server. The download area requires free registration and provides the latest versions of all NeptuneLabs FSI applications.

https://jersey.java.net: Jersey is the JAX-RS implementation used by FSI Server to provide the part of the API that allows modifying data on the server.

http://jettison.codehaus.org: Java API used by Jersey to read and write JSON.

http://www.freemarker.org: The template engine used to render Info and List requests.

B. Changes in the OpenAPI

To make the transition from previous version of FSI Server to FSI Server 3 as easy as possible for third party applications, the changes in the OpenAPI were reduced to the necessary minimum. Most custom code should run without requiring modifications. In detail, the changes where client code might be affected are as follows:

The login process has been extended to support plain text password required for server side Kerberos authentication. If your FSI Server does use Kerberos authentication, the client application will have to be modified the bahve as stated in section 3.2.

The designated service user and the status request for monitoring FSI Server have been dropped in favour of a JMX interface. Detailed information on the JMX interface is available in the FSI Server manual.

In addition to the above mentioned some parameternames have been deprecated. They are still supported for compatibility reasons, but should be changed in code generated after the FSI Server 3 release date.

In the requests directed to the FSI Server for image or metadata retrieval the old parameter *profile* is replaced by the new *renderer*.

Furthermore the parameter *tpl* has been replaced by template for *readability*.

As FSI Server 3 supports all kind of digital assets and no longer only images, the service requests to manage files on the server have changed from *.../service/image/...* to *.../service/file/...* .

Index

OpenAPI for FSI Server

NeptuneLabs GmbH

Lagesche Str. 32
D-32657 Lemgo
Germany

Fon: +49 (0) 5261-28732-0
Fax: +49 (0) 5261-28732-29
eMail: info@neptunelabs.com
WWW: www.neptunelabs.com
www.fsi-viewer.com

www.fsi-viewer.com

Neptune Labs

Internet Application Development

NeptuneLabs GmbH
Lagesche Str. 32
32657 Lemgo
Germany
fon: + 49 5261 - 28732- 0
fax: + 49 5261 - 28732- 29
info@neptunelabs.com
www.neptunelabs.com