https://doi.org/10.2352/issn.2168-3204.2017.1.0.28
© 2017; Society for Imaging Science and Technology

Simple Image Presentation Interface (SIPI) – an IIIF-based Image-Server

Lukas Rosenthaler, Peter Fornaro, Andrea Bianco and Benjamin Geer; Digital Humanities Lab, University of Basel; Basel; Switzerland

Abstract

The International Image Interoperability Framework (IIIF) [1] is a widely accepted and fast growing standard to present images as web-resources. The IIIF-standard defines an URL-syntax to access, transform and reformat the desired image. An IIIF-server converts the image on-the-fly based on the desired parameters and transfers the image using the HTTP protocol to the client. We designed and implemented an advanced, extremely flexible, fully IIIF compliant server in C++11 offering advanced features that go beyond the IIIF standard. Due to its flexibility, can easily be integrated into existing environments and thus facilitates the transformation of existing archiving platforms to support the IIIF protocol.

Introduction: Images and the Internet

On Feb. 26th 1993, Marc Andreessen, co-author of *Mosaic* (the first widely used Web browser) and co-founder of Netscape, proposed in a message to a mailing list a new HTML-tag *IMG* with a mandatory attribute *src="url"* as follows:

> *This names a bitmap or pixmap file for the browser to attempt to pull over the network and interpret as an image, to be embedded in the text at the point of the tag's occurrence.*
> *Browsers should be afforded flexibility as to which image formats they support. Xbm and Xpm are good ones to support, for example. If a browser cannot interpret a given format, it can do whatever it wants instead (X Mosaic will pop up a default bitmap as a placeholder).*
> [2]

This proposal which found wide acceptance allowed for the first time to display images in a web browser. Since then, images have become an indispensable part of the internet. The revolutionary idea of Andereessen has been to address an image like any other resource in the internet using an URL. It's up to the web client (e.g. the browser) to fetch the image and display it. The HTTP-protocol header *Content-type* allows the server to send the MIME-type in order to allow the client to render the image properly. Soon after the introduction of the <img>tag, support for the GIF-format[5] for graphic-like images and the JPEG-format for photographic images has been implemented in most browser. Since JPEG allows an efficient (but lossy) compression of image data, it was especially well suited for high resolution photographic images, since the data compression reduced the loading times over the – at that time rather slow – internet connections. It is to note that the URL within an <img>tag usually only references a specific image as a static resource with a given resolution, format and quality. After the image has been loaded completely, modern browser allow some scaling and – using tricky HTML and CSS code – some cropping of the loaded image.

Nevertheless, images became an integral part of the internet and soon special purpose web-based image databases began to surface. At the same time, the progress of digital image capture technology started to rival and surpass analogue photography in ease of handling and image quality. Today digital photography has almost completely replaced analogue photography in museums and archives. The processed (e.g. cropped, contrast and color corrected) high resolution, high quality digital images have to be considered *digital master copies* that have to be preserved with longevity in mind.

Given the limitations of the image tag and the typical web-server/web-browser functionality, it became necessary for image databases to store and manage several derivatives of the master copy representing different resolutions, qualities and possibly formats (usually just the JPEG format is accessible on the web). This makes web based image databases rather complex, inflexible and requires the storage of multiple copies of the same image in different representations. In addition the user has only access to a limited, predefined set of image variants.

The International Image Interoperability Framework IIIF

Digital images have become an important part in research both in science as well as in the humanities. While in science most image interpretation is performed by computational methods such as statistical image classification, pattern recognition algorithms etc., the nature of digital images in humanities is quite different. Many images in the humanities depict cultural artifacts such as paintings, sculptures, documentary photographs etc.. Another significant class of digital images in the humanities represent written texts, especially manuscripts. Scholars working with ancient papyri or medieval codices, but also those working on critical editions in literature or science depend on access to the original manuscripts. Sometimes these original texts have been taken apart and the fragments are dispersed all over the world – as it is especially the case for some medieval manuscripts. Many of these fragments have been digitized and are available on the internet. However, there has been a strong need to have a *standard* way to access these images and make them available for research in a very flexible way. Out of this community the IIIF emerged as a standardized way to address images or parts thereof using a flexible URL syntax. This allows to compare and assemble images from different places in the world in a common way and to develop new research tools.

Thus, the goal of the IIIF standard is to enable the sharing of digital images using the Linked Open Data (LOD) principles

in the field of cultural heritage humanities research. It has been conceived by a large consortium of serious players in the field of cultural heritage and research institutions such as ARTstore, the British Library, Harvard University, to name a few. There are also many web-based viewers available that are capable of displaying high-resolution images using tiling that is supported by the IIIF server.

IIIF Standard

The IIIF standard defines a standardized API on how to address images using the well known URL syntax. In addition, it also provides a standardized way to access essential metadata about digital images. The basic syntax to access an image itself is as follows:

```
{scheme}://{server}{/prefix}/{identifier}/
   {region}/{size}/{rotation}/{quality}.{format}
```

where {*scheme*} is either "http" or "https", {*server*} the DNS name of the service, {*prefix*} a service-specific name which can be used to order collections etc., {*identifier*} the image identifier. The IIIF-standard does not impose a special formatting of the image identifier. It may be a number, an archiving signature, a filename etc. {*region*} allows to select a rectangular region (e.g. "region of interest") of the image. {*size*} allows to select the size (in pixels or percentage) that the selected region should be adjusted to. The {*rotation*} parameter allows to give an arbitrary rotation and/or mirroring of the selected part of the image. For both {*region*} and {*size*} the keyword "full" indicates to transmit the full image at full resolution. {*quality*}allows to select B/W, gray and color rendering, while {*format*} indicates the file format that should be used. Currently the standard includes JPEG (.jpg), TIFF (.tif), PNG (.png), GIF (.gif), JPEG2000 (.jp2), PDF (.pdf) and WEBP (.webp).

The server dynamically responds to such a request, prepares the image from a master image according to the specification given on the URL and sends the data using the HTTP protocol. The IIIF standard does not imply a specific implementation of the server or on how the different derivative images are being generated. There are several servers available that either natively support the IIIF V2.0 standard or offer some protocol translation tools[1]. However, most of these image servers do have some serious drawbacks.

IIIF and JPEG2000

Since there are almost unlimited variants of an image that can be requested using the IIIF-syntax, a IIIF compliant image server has to be able to calculate "on the fly" the required variant. Therefore IIIF servers typically store only one high resolution, high quality (master-) image from which the required image variant is dynamically derived. It is obvious that these image transformations have to be performed with very high efficiency in order to guarantee acceptable response times. Using the JPEG2000 format as server-side master format has several crucial advantages compared to other image formats:

- it has a lossless compression mode and supports images with 16 Bit depth. Thus it can store true master images.
- using internally a resolution pyramid, access to lower resolutions is highly efficient. Given proper compression parameters, eventually only a fraction of the (usually quite big) master file has to read.
- JPEG2000 supports tiling which makes the selection of region of interests very efficient.

Therefore most IIIF servers support JPEG2000 as master image file format. However, software support for JPEG2000 is quite hard to find. The few open source implementations of JPEG2000 available do have performance problems. Some IIIF servers rely on the binary distribution of a compression/decompression tool by kakadu-software [4]. While these tools offers a wide variety of compression parameters and options, support for other features such as metadata, different file formats etc. is rather poor. However, the code is highly efficient and even supports multithreading.

Why is "yet another IIIF server" needed?

By a mandate of the State Secretariat for Education, Research and Innovation (SERI), the Swiss Academy of Humanities and Social Sciences (SAHSS) has created a new institution for the preservation and long-term curation of research data in the humanities, the "Data and Service Center for the Humanities" (DaSCH)[2]. It is insuring permanent access to research data in order to make it available for further research. A pilot started in 2013 and has been successfully finished. The DaSCH has been permanently installed on January 1st 2017. The Digital Humanities Lab (DHLab)[3] of the University of Basel has been mandated with the operation of the new institution. For this purpose, it developed a flexible research platform based on semantic web technologies (RDF, RDFS, OWL) [3]. Besides text sources, about 500'000 high-resolution images have been ingested to the system during the pilot phase. Together with annotations, internal and external linkage etc. this results in approx. 50 Mio. RDF-triples. We decided recently to use IIIF for presenting the images in order to maximize the interoperability with external systems. Furthermore, we need to preserve only one image file, since IIIF allows using the archiving master also for dissemination and presentation.

However none of the existing IIIF-compliant servers satisfied our demanding requirements that are:

- Interoperability with external databases (e.g. the RDF-triplestore) containing annotations, metadata etc. as well as access permissions.
- Preservation of all embedded metadata (e.g. EXIF, IPTC, XMP, TIFF etc.) during all format conversions.
- ICC color profile conversions where necessary.
- User authentication using JSON Web-Tokens (JWT) or a similar scheme that is compatible with current IIIF standard for authentication [6]
- High-performance transformation of images including rotation, format conversions for 16 bit and 8 bit images.
- Support of Secure Socket Layer (SSL/https).
- Configurable image cache in order to reduce the computational load on the server.
- Support of cross origin resource sharing (CORS)

[1] The IIIF website lists some of the most prominent compliant servers. See `http://iiif.io/apps-demos/#image-servers`

[2] see `dasch.swiss`

[3] See `http://dhlab.unibas.ch`

- Import and transformation of images. The server must be able to import (upload) images and convert them to the desired master file format (in our case JPEG2000).
- Features beyond the scope of the IIIF-standard such as adding watermarks, size restrictions etc.
- Integrated simple web server functionality
- modular extensibility, e.g. integrating support for RTI imaging (both initial transformation and serving a web-based RTI-viewer) [7] [8]

None of the existing IIIF compatible image servers comes close to these requirements, so we decided to create our own, fully IIIF compliant image server.

Implementation of the Simple Image Presentation Interface SIPI

In a first step, some fundamental decisions regarding the implementation had to be taken. Our starting point was as follows

SIPI Design Principles

- SIPI must run primarily on Linux and related systems (e.g. BSD Unix, MacOS).
- The basic functionality (image decoding and encoding, image manipulation, HTTP server etc.) has to be implemented using a compiled language in order to get the required performance. We decided to use C++11 as basic implementation language
- In order to allow a flexible configuration and adaption to different ecosystems, a script language should to be embedded. We decided to use Lua [9] which is widely used in the gaming world (e.g. World of Warcraft) because of its small footprint and high performance. Lua is especially designed to be embedded into other applications.
- The server should store one master file using lossless JPEG2000 compression. Since the available open source JPEG2000 libraries are not satisfying regarding performance, we decided to use the – unfortunately not free – kakadu implementation of JPEG2000 [4]. Kakadu distributes the full source code to the licensees and allows the binary distribution of programs using the kakadu code (given the proper license), but it prohibits the distribution of the kakadu library itself as binary file[4].
- SIPI should be made available as open source based on the GNU Affero General Public License 3.0 [10]. The code is to be publicly available on github.com (`https://github.com/dhlab-basel/Sipi`).

SIPI Architecture

SIPI is written in C++11 und thus requires at least gcc 5.3 or clang 8.0. As build-system we decided to rely on Cmake (version 3.0 or later) in order to make the building to some degree independent of the used linux variant. SIPI depends on several third party (open source) libraries. With some noted exception, specific versions are downloaded and compiled locally during the build process, but not installed globally. Since some library versions may be in conflict with already installed libraries, we decided to prefer static compilation which does not require the installation of shared libraries. In order to read and write different file formats, we are using the following libraries:

- TIFF: libtiff-4.0.7
- JPEG: libjpeg-v9b
- PNG: libpng-1.6.27
- JPEG2000: kakadu-7.9[5]

In order to interpret the various metadata formats, the following libraries are used:

- EXIF, IPTC, XMP: exiv2-0.25
- ICC: lcms2-2.8 (little CMS V2)

Other important libraries used are:

- Lua interpreter: lua-5.3.1
- Sqlite3: sqlite-autoconf-3140200
- Curl: curl-7.51.0

Some other helper libraries are also used. All these libraries are downloaded and compiled locally during the cmake process without global installation on the system. There are only a few exceptions, most notably *OpenSLL* that must be installed systemwide (for more details refer to the README of SIPI).

The generic architecture is shown in fig. 1. SIPI makes heavy use of predefined C++11 classes.

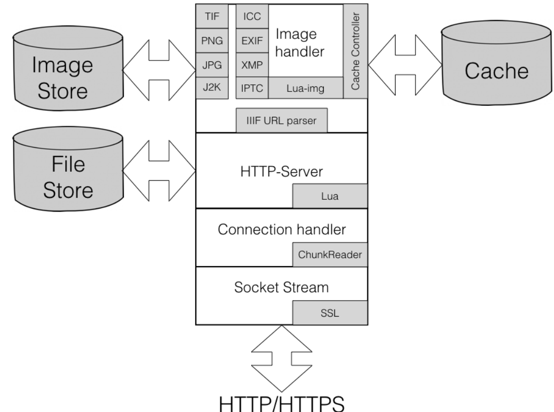

Figure 1. *Sipi Architecture.*

Socket Stream Layer

At the bottom, the socket communication (HTTP connection) is implemented as a subclass of the C++11 class *sockstream* that includes dynamic buffering and optional OpenSLL communication.

Connection Handler

The connection handler deals with the HTTP protocol including chunked transfer. It parses incoming HTTP headers,

[4]However the license price tag is very reasonable for public institutions so we do not regard this restriction as a major hurdle for using SIPI in other institutions

[5]Please note that this library is unfortunately not open source and has to be licensed!

 © 2017 SOCIETY FOR IMAGING SCIENCE AND TECHNOLOGY

HTTP message bodies (e.g. multipart/form-data or application/x-www-form-urlencoded formatted bodies) and prepares the appropriate data structures. It also assembles and formats outgoing HTTP messages. To the upper layers it looks like a simple stream interface and thus hides the complexity of the HTTP protocol completely. This feature is especially important in order to allow the image libraries to use the HTTP connection as simple stream.

Server Layer

Simple HTTP Server The server layer implements the basic control of a web server. It uses multithreading with the maximal number of concurrent threads limited in the configuration file. The server is also responsible for implementing the "keep-alive" feature of the HTTP protocol. While each request creates a new instance of the connection handler, the socket itself may be kept open according to the keep-alive requirements. The server layer also creates for each request a new instance of the Lua interpreter. Having for each request a new instance of the Lua interpreter eliminates the risk of security breaches within Lua (e.g. data remaining accessible between requests). Since the Lua interpreter is very small and efficient, the instantiation of a new Lua interpreter for each request is justifiable and does not degrade performance significantly. The embedded Lua is enhanced with some special commands, e.g. to read and send data using the HTTP connection, RESTful API's etc. (see SIPI documentation). Using the curl library, Lua is also able to act as a HTTP client and query other servers. This enables SIPI to communicate with existing frameworks given the offer a (possibly RESTful) API. Finally the server layer implements a simple but complete, small footprint web server which is able to transmit static files through user-definable routes. In addition dynamic content can be created using Lua-code embedded in HTML. A file with the extension ".elua" may contain arbitrary Lua code inbetween <lua>?</lua>tags.

The HTTP server also supports HTTP POST requests with "multipart/form-data", if an appropriate route is configured. Thus image uploads are possible using only the SIPI. Authentication etc. of uploads are provided using Lua.

IIIF Pre-Flight Script If the server receives IIIF compliant request, it will first execute a site specific Lua script (defined in the configuration) which must either return "allow", "restrict" or "deny". In case of "allow", it also returns the path to the master file, which is then sent as requested. If the pre-flight script returns "restrict", it must, in addition to the path to the master file, also return either a path to a watermark file or a maximum image dimension. Then either the watermark is applied to the image file or the image size is restricted to the given value. SIPI-Lua and the server layer also implement JSON web tokens (JWT) [12] for authentication. Thus, during the pre-flight execution, JSON web tokens may be examined, databases may be queried (e.g. Sqlite3 is embedded into SIPI) or external frameworks may be queried using HTTP client functions provided by SIPI-Lua. Thus a full integration into existing frameworks is possible and SIPI can be used to transform legacy image databases to IIIF conformance. The server layers also implements CORS[6] to facilitate the integration into complex environments.

[6]Cross-origin resource sharing (CORS) is a mechanism that allows resources on a web page to be requested from another domain outside the domain from which the first resource was served.

Image Layer

The image handler implements all the necessary routines for image handling. It interprets the IIIF specific parts of the URL, reads and writes the different file formats from/to disk and the HTTP connection, manipulates size, region of interest, bit depth, ICC profiles etc. and is able to transfer the metadata between the different image formats. Internally the images as well as the different metadata are represented in a file format agnostic way. Formatting and embedding of the many kinds metadata into the different file formats has been a major hurdle because each file format treats metadata completely different and information about it is quite hard to get.

Some file formats (e.g. the TIFF format) do not allow the streaming of the data, since they rely on random access to the data while reading and writing. In these (and only these) cases, a memory file is being created and the file is completely written to memory before sending it to the HTTP connection.

Caching

Since the decoding and encoding of the different file formats as well as the image manipulations are time and CPU consuming, a simple image cache has been implemented. It is to note there are may be many different IIIF compliant URL's which will result in the same image to be sent. The IIIF standard defines a *canonical URL* which uniquely identifies each image variant. We are using this canonical URL (which is derived for each request) as index into a simple map of cached files. In order to determine the use of the cached file, also the creation times of the original file and the cached version is being considered.

If a file is not in the cache or has to be replaced, the cache file stream is created and handed over to the connection layer which is responsible to send the data both to the HTTP connection and the cache file stream (und thus writing the cache filed). Using this mechanism, a high efficiency and low latency is guaranteed, since sending the data to the HTTP connection and the disk file are being done at the same time. The size and maximal number of files kept in the cache can freely be configured. File deletion is prioritized by access time. This the file that has not been accessed for the longest time will be the next to be removed from the cache.

Caching improves the performance of a IIIF based server significantly. This holds especially true if tiled viewers such as openseadragon[7] or mirador[8] are being used on the client side.

Command Line Interface

The server image can also be used as a command line tool to convert, analyze and manipulate images. Besides file format conversions, simple ICC profile manipulations, bit depth conversions etc. are possible.

Configuration

The many options and configuration parameters are defined using a Lua configuration script which basically conststs of a first lua object containing the configuration options and of a second lua object defining the routes. Lua allows a very flexible configura-

[7]See `https://openseadragon.github.io`
[8]See `http://projectmirador.org`

tion since the configuration file is in fact an executable Lua script with all possibilities given offered by Lua.

Roadmap of SIPI development

In the future SIPI will be extended in several directions.

- We would like to extend SIPI also for providing access to moving image (video, film).
- RTI imaging[9] will be integrated into SIPI.
- We will also extend SIPI to better support the IIIF presentation API and the IIIF search API, given that SIPI is integrated into some kind of database environment.
- We also plan to extend SIPI in a way that digital facsimiles of manuscripts and there transcription based on TEI [13] can be integrated.

We hope to build a string open source community around SIPI which will further enhance the applications and add new features.

Use Case

We are currently using SIPI in a productive environment for the Data and Service Center for the Humanities (DaSCH). The DaSCH has two goals: On one hand it preserves the long term accessibility to research data from the humanities long beyond the end of a given research project. For example, the results of a critical edition may still be relevant and important many decades after the edition project itself endet. On the other hand the DaSCH will offer a virtual research environment including long term repository to scholars currently working on research projects. In order to comply with these two goals, we built an infrastructure as shown in fig. 2. Semantic web technologies (RDF, OWL etc.)

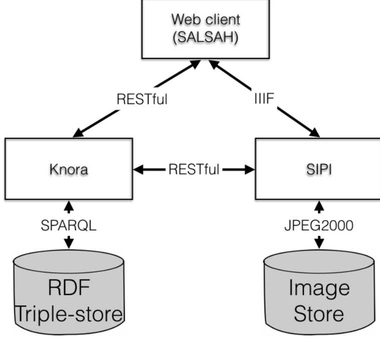

Figure 2. *DaSCH Architecture.*

are a highly flexible way of representing complex linked data. In fact, it is allows to simulate virtually all data models (e.g. relational databases, graph databases, key-value stores etc.) using a single data representation (triple store). However, it does not make sense to store binary digital objects directly within such a database. Therefore we decided to use an (extended) IIIF based server for serving binary objects. Due to reasons of copyright, personal rights etc., not all resources are available as open access data. Therefore we had to implement some form of user authentication. The web-based formt end of the virtual research system uses the RESTful API to interact with the knora base system. The front end then may demand resources from the SIPI server which either gets the access permissions using a JSON web token or by querying itself the knora[10] middleware.

Currently we host about 25 research projects within our framework, from the Bernoulli-Euler Online edition to the Lexicon Iconographicum Mythologi Antic (`http://www.weblimc.org/search`) to the online foto archive of the museum of modern art "Kunsthalle" (`http://www.salsah.org/kuhaba/`. Currently we are migrating the images to the new SIPI server. To the end of the year, we expect to host about >0.5 Mio images using SIPI.

Conclusion

With SIPI we implemented a versatile, high performance IIIF image server which implements many features that go beyond the scope of other IIIF compliant servers. Its rich set of features make it especially well suited to be integrated into complex environments. It's key features are:

- Fully IIIF compliant
- High performance C++11 implementation
- Preservation of EXIF, IPTC, XMP and TIFF metadata
- Support for ICC color profiles (including some color profile conversions)
- Support for images with 16 Bit depth
- IIIF compliant authentication using JSON web tokens
- Embedded Lua interpreter and execution of pre-flight script
- Embedded simple web server (incl. file uploads)
- Embedded Sqlite3 database access
- Integrated command line interface for file format conversions

SIPI can be used to create simple standalone image databases or it can be integrated into complex frameworks and interact with existing databases and external applications. In all cases is provides a high performance IIIF conforming access to digital images. SIPI is currently extensively using within the DaSCH platform.

References

[1] See `http://iiif.io`
[2] See `http://1997.webhistory.org/www.lists/www-talk.1993q1/0182.html`
[3] See `https://www.w3.org/standards/semanticweb/`
[4] See `http://kakadusoftware.com`
[5] See `https://www.w3.org/Graphics/GIF/spec-gif87.txt`
[6] See `http://iiif.io/api/auth/1.0/`
[7] Peter Fornaro, Andrea Bianco and Lukas Rosenthaler, Digital Materiality with Enhanced Reflectance Transformation Imaging. Archiving Conference. 19. April 2016 Vol. 2016, no. 1, p. 11-14
[8] Peter Fornaro, Andrea Bianco, Aeneas Kaiser and Lukas Rosenthaler, Enhanced RTI for gloss reproduction, in Electronic Imaging, 8 (2017), pp. 66-72, `https://doi.org/10.2352/ISSN.2470-1173.2017.8.MAAP-284`
[9] See `Seehttp://www.lua.org`

[9] See `https://en.wikipedia.org/wiki/Polynomial_texture_mapping`

[10] see `http://www.knora.org`

© 2017 SOCIETY FOR IMAGING SCIENCE AND TECHNOLOGY

[10] See https://www.gnu.org/licenses/agpl-3.0.en.html
[11] See https://cmake.org
[12] See https://en.wikipedia.org/wiki/JSON_Web_Token
[13] See http://www.tei-c.org/index.xml

Author Biography

Lukas Rosenthaler is part of the management team of the Digital Humanities Lab of the University of Basel and has a background in Physics and Astronomie. His research is focused on virtual research environments for the Humanities, data modeling using sematic web technologies, image processing and computer vision. He is the head of the national "Data and Service Center for the Humanities".

Peter Fornaro is part of the management team of the Digital Humanities Lab of the University of Basel and has a background in Physics, Electronic Engineering and Photography. He is doing research in the field of digital archiving, imaging, cultural heritage preservation and computational photography. Fornaro is also a member of the Swiss Commission for Cultural Heritage Preservation (EKKGS).

Andrea Bianco is a PhD student in experimental physics at the Digital Humanities Lab of the University of Basel. His research is focused on digital image processing, spacing from hardware to software. He is enthusiastically involved in development process of SIPI.

Benjamin Geer is part of the development team of the DaSCH infrastructure. He is a specialist in Middle Eastern studies and Computer Science. His focus is on semantic web technologies (ontologies, SPARQL etc.), data structures and software development in scala and C++.