# Library Requirements Document

Marcel Timmerman <mt1957@gmail.com>



## Table of Contents



# 1. Introduction

Long ago I've started to think about how to maintain the large amount of documents on my system which consists of anything textual or other kinds of imagery like photos, films, music and books. Storing on disk you can give it a name and store in some place in the hope that you can find it later. You can than also make links to the same data to provide extra information in the form of a filename. This will soon be a problem when the file is placed in another location.

I now wanted a solution which does not duplicate any information except for a backup and also not clobbering the filesystem with links to some other location. And there this program comes in.

In this document I try to make clear what is needed to build the modules and program and what to build myself. This must be viewed in light of a set of prioritized requirements.

# 2. Ideas to implement

Here a list of thoughts will show what I like to include in this system. It does not mention if this will be feasable or not. That will come in a later part where things like dependencies will be investigated.

## 2.1. Files can be located anywhere

This system will not manage documents. It will manage metadata about the documents. So it is not nessesary to store them somewhere. It is however nice when it can detect duplicates when another document is entered by the user. This duplication can be caused by backups or archives.

A file can only be found on the local disk or externally connected disk. There are however other places where documents can be found such as network attached storage (NAS), media stores or on webservers. In the same process as adding the metadata to the database these remote documents might be copied to local store to prevent dissapearing from the net when you find it still important.

## 2.2. Not only files

A file on a disk is pointed to by a name and path. There are other ways to get to a document like using a unified resource locator (URL).

### 2.2.1. Use of mimetypes and document suffixes

Mimetypes are an important type of description method to show what can be done with the document. The list can also be used to start native applications to process a particular document. According to their mimetype of the document it mostly has also a proper suffix such as *.txt* or *.html*. See also [MIMETYPES].

A few examples are

- *text/plain:* This is simple text format mostly created with simple text editors.
- *audio/mpeg3:* A type of audio file with document suffix of *mp3*.

### 2.2.2. Use of protocols

Protocols are used to get to the document before processing it. E.g. the *http* protocol is used to get a webpage from a site on the network and *file* is used often to get a document from the local filesystem. See also [MIMETYPES].

The following list is a series of protocols which might be supported.

- *file:* Protocol to get documents from a filesystem.
- *http* and *https:* Protocols to get webpage documents from a web server.
- *ftp:* File transfer protocol.

## 2.3. Metadata

What information do we want to store. Well, it must be something about the document. Furthermore, keys under which a document might be catagorized. A description will sometimes help. There are on the internet numerous descriptions available of known items like books and movies. Besides the list below, users must be capable to add new metadat attributes.

- *name*, Unique name
- *path*, complete and absolute path to the document
- *uri*, url or uri
- *description*, a description
- *author*, a set of data like name and surname, address and email etc
- *datetime*, Date and time of retrieval, date and time of modification or current date and time

# 3. Implementation

This software package should come with several modules and programs to suit several ways of accessing the data. There is also an issue of making the software platform independent so everyone can be happy with it.

- *The programming language*

  The first item to think about is the choice of programming language. A scripting language would be a proper choice because these languages have a higher level of coding and will access the underlaying system in a platform independent way. The language I want to choose is perl6. Yes, the still unfinished perl version. I am very confident that the language gets its first release this year(2015) and wanted to learn about the language by doing this project.

The second approach is to use a browser to do the work. There we can use *html5*, *css3* and *javascript* and libraries. There is also a server side scripting which can be any of *perl6*, *perl 5* or javascript *(nodejs)*

- *The storage method*
- *Storage*

  The name of the database and the names of the collections

# 4. Dependencies

The program will be depending on several modules and programs. That is only logical because we do not want to reinvent the wheel(s) again do we? We only try not to select those software which will bind it to some platform as explained above.

- *perl6:* The followup version of perl 5.*. The program is not yet completely finished but will be soon (2015-01). This program is a interpreter/compiler which can compile the script into some intermediary
- *perl6:* The followup version of perl 5.*. The program is not yet completely finished but will be soon (2015-01).

# 5. Priorities

# Bibliography

[MIMETYPES] *A list of all mimetypes*. internet [http://www.sitepoint.com/web-foundations/mime-types-complete-list/]