Improving Photoshop PSD file format support in the free software image editor GIMP

Rocafort Ferrer, Guiu
Curs 2016 - 2017

Director: Joan Codina
GRAU EN ENGINYERIA INFORMÀTICA

Treball de Fi de Grau
Dedicated to Marina,

for all the support in this struggling part of my life.
Acknowledgments

I would like to acknowledge Michael Narreter (@mitch), for his advice and help provided in the GIMP IRC Channel.

I would also like to acknowledge my tutor, Joan Codina, for this understanding and helpful advice during the elaboration of this work.

Thanks for reading. Thanks for your time.
Summary

This project aims to improve the support for Photoshop PSD files in the free software image editor GIMP. We first analyzes the development environment in which we develop. Then we talk about the current code and check the existing features.

We then find that the actual code is not well designed and very difficult to maintain, therefore decision of developing a new code from the start is then made. We empathize how important is to develop a good design for a software to be able to maintain it and allow it to develop through time.

We design a new code structure which is more modular, and has integration / unitary tests. Following the design, the code is developed and the features are implemented.

Resum

Aquest projecte té com a objectiu millorar el suport dels fitxers de Photoshop PSD a l’editor de programari lliure GIMP. Primerament analitzem el entorn de desenvolupament en el que programes. Després, parlem del codi existent i comprovem les característiques existents.

Ens trobem amb un codi mal dissenyat, i molt difícil de mantenir, per tant, es pren la decisió de dissenyar un codi amb una estructura nova, més modular, que permeti aplicar tests unitaris i d’integració amb GIMP. Remarquem que aquestes característiques son essencials per a un codi que es pugui mantenir fàcilment i pugui créixer i adaptar-se al futur.

Dissenyem una nova estructura de codi que és més modular, que permet fer tests unitaris de cada part. Finalment, el codi és desenvolupat i les característiques son implementades.
## CONTENTS

1 **INTRODUCTION** ................................................................. 13
   1.1 Motivations .................................................................. 13
   1.2 Objectives ................................................................ 13
   1.3 Outline ...................................................................... 14

2 **CONTEXT** ....................................................................... 15
   2.1 Community .............................................................. 15
   2.2 Development status .................................................. 15
   2.3 Source organization ................................................... 15
   2.4 Parasites .................................................................... 16
   2.5 Plugins ...................................................................... 16

3 **PSD COMPATIBILITY** ....................................................... 18
   3.1 Definition .................................................................. 18
   3.2 Format of the graphics ............................................... 19
   3.3 PSD Import ............................................................... 19
   3.4 PSD export ............................................................... 31

4 **EXISTING PSD PLUGIN** ..................................................... 32
   4.1 Import code .............................................................. 32
   4.2 Thumbnail .................................................................. 33
   4.3 Export code .............................................................. 33
   4.4 Open bugs regarding PSD import / export .................... 34

5 **PLANNING** ..................................................................... 36
   5.1 Approach .................................................................. 36
   5.2 The Minimal PSD file ............................................... 36
   5.3 Main Data structures .................................................. 37
   5.4 Code structure .......................................................... 37
   5.5 Exception management .............................................. 38
   5.6 Export User Interface ................................................ 38
   5.7 Testing framework .................................................... 40
   5.8 File parsing strategies ................................................ 41
   5.9 Repository management ............................................ 42
   5.10 Tasks ..................................................................... 43
   5.11 Temporal planning ................................................... 43

6 **EXECUTION** .................................................................... 45
   6.1 Setting up the development environment ..................... 45
   6.2 Build empty functional plugin ..................................... 46
   6.3 Create empty code structure ....................................... 47
   6.4 Prepare code for unit tests .......................................... 48
6.5 Babl bug fixed................................................................. 48
6.6 Develop support for the minimal PSD file.................. 48
6.7 Add support for rest of color modes.......................... 49
6.8 Add integration tests.................................................. 49

7 RESULTS.............................................................................. 49
  7.1 Future work.............................................................. 50
  7.2 Conclusions............................................................. 50

8 GLOSSARY OF TERMS......................................................... 52

9 BIBLIOGRAPHIC REFERENCES........................................... 53
LIST OF FIGURES

Illustration 1: GIMP plugin architecture.........................................................17
Illustration 2: Example graphic describing notation........................................19
Illustration 3: PSD high level structure..........................................................20
Illustration 4: File header fields....................................................................20
Illustration 5: Color mode data fields.............................................................21
Illustration 6: Layer and mask section............................................................26
Illustration 7: Layers and Masks Information................................................26
Illustration 8: Layer information subsection, Layer records.............................26
Illustration 9: Layer records subsection, Layer mask data...............................28
Illustration 10: Layer records subsection, Layer blending ranges.....................28
Illustration 11: Layer channel image data.......................................................28
Illustration 12: Global layer mask info............................................................28
Illustration 13: Additional layer information..................................................28
Illustration 14: Image data section..................................................................31
Drawing 1: Minimal PSD file structure............................................................36
Illustration 15: PSD Export dialog.................................................................39
Drawing 2: Tasks temporal planning...............................................................44
LIST OF TABLES

Tabla 1: Image resources IDs compatibility.................................................................25
Table 2: PSD layer blend modes compatibility.............................................................27
Tabla 3: Layer and mask additional resources.............................................................31
Tabla 4: List of open bugs regarding PSD import / export............................................35
Tabla 5: Available color modes from GIMP images.......................................................39
Tabla 6: Available pixel depths per each color mode..................................................39
Tabla 7: Higher pixel depth transformations...............................................................40
1 Introduction

1.1 Motivations
For many years, the most used image editing software have been the privative software Adobe Photoshop. Its first version was released in 1988 and soon became the main standard for graphic industry, until nowadays. Adobe Photoshop saves projects in an open file format called PSD, making the PSD file format very common and ubiquitous. [3]

Consequently, there is many Photoshop material available in the web. If you work professionally in photo editing or design you need to deal with PSD files in on everyday basis.

GIMP (GNU Image Manipulation System) [4] is an open source alternative to Adobe Photoshop, but many professional designers and photo editors can’t use it because it’s very partial and incomplete support for the PSD file format. A better support for this format would allow many people to use GIMP and improve greatly the interoperativity between those software.

My main motivation is to improve GIMP support for the PSD format and therefore promote free software software for photo editing.

My second motivation is to create something which will have a practical and real use in the world. I’ve been using free software operating systems for many years, and i benefited from the work of the community. I want to give back to the free software environment by improving an existing software, so others can benefit from my work.

1.2 Objectives
The main objective of this work is to improve GIMP compatibility with the PSD file format. This roughly involves:

- Identification and familiarization with GIMP community and source code
- First approach to current support for PSD files in GIMP
- Finding incompatibilities between GIMP and the PSD format and vice versa
- Planning how to manage incompatibilities
- Plan the improvement of the PSD support
- Perform the development tasks
- Get the code reviewed by the GIMP community and merged into the GIMP main repository

1.3 Outline

In Context, we discuss the environment in which we are going to develop, GIMP community, and the internal details of GIMP code and its plugin architecture.

In PSD compatibility We define the idea of compatibility, and we examine all the components in the PSD format. For each feature we decide if it is importable and exportable from GIMP and the possible transformations needed to do so.

In Existing plugin, we analyze the existent code from GIMP repository. We visually inspect the code and we understand the status and maturity of the code.

In Planning, we design a new software architecture, and plan how to improve the number of imported / exported features given the existing code.

In Execution, we describe the development of the tasks and the unexpected problems encountered during the process.

Finally, in Results, we present the achieved results, and talk about next steps and further work that could be done in the future.
2 Context

2.1 Community
GIMP is an open source image editing software available in Linux, Windows and Mac OS operating systems. The GIMP project has a very established community with more than 550 different contributors and around 800.000 lines of code [5]. The project started in 1997, so now is about 20 years old.

The main communication methods in the community are GIMP developer mail list [6] and the IRC channel. The bug managing is done using the Bugzilla platform [7].

2.2 Development status
GIMP source code is mainly written in C and relies on many libraries to provide its functionalities. It relies heavily on Glib library (low level C multi platform library) [8]. The user interface is managed by GTK [9]. The rendering of vector content is managed by Cairo [10]. The rendering of text is managed by Pango [11]. The source code is hosted in the Gnome git server [12].

In the last years, part of GIMP source code has been divided into independent libraries called Gegl [13] and Babl [14]. Those changes are intended to give GIMP full support for non-destructive image editing. Gegl gives the non-destructive capabilities defining an image as a tree of operations and transformations. Babl is a support library used by Gegl for pixel format conversion. This is a work in progress and the conversion of GIMP code to use Gegl library is a work in progress [15].

The current stable version of the software is 2.8.22. The upcoming stable release 2.10 will include the following improvements which are in development at the time of writing.

- Full GTK+ 3 migration from GTK+ 2.
- Full transition to Gegl.

2.3 Source organization
GIMP code is separated in different static libraries and the main core app. The main app, with the GUI management and procedure registration code is contained in the app folder.

Then, there are several libraries that register procedures and interact with the main core and provide most of the functionalities. Those are libgimp, libgimpbase, libgimpcolor, libgimpconfig, libgimpmath, libgimpmodule, libgimpthumb, and libgimpwidgets.
Inside the libgimp folder, the PDB accessible procedures are defined. Each code file contains a ‘filename_pdb.c’ and ‘filename_pdb.h’ files defining the accessible methods through the PDB.

For example, ‘libgimp/gimpimage_pdb.h’ contains all the public methods for manipulating images.

2.4 Parasites

In a GIMP image, a parasite is an arbitrary chunk of data that can be attached to the image. Parasites are a way to maintain data contained in some external file format that GIMP does not use or recognize. This data is can be written back if the file is exported in the original format [16] [17].

The parasite definition makes use a void C pointer to hold any information. Therefore any arbitrary information can be stored, and it is up to the plugin to manage the encoding and decoding of the data [18]. Gimp is completely unaware of the information stored inside the parasite. Here we provide the definition of the parasite struct. (gpointer is a void* pointer data type provided by Glib) [19].

```
struct _GimpParasite
{
    gchar    *name;  /* The name of the parasite. USE A UNIQUE PREFIX! */
    guint32  flags;  /* save Parasite in XCF file, etc. */
    guint32  size;   /* amount of data */
    gpointer data;   /* a pointer to the data. plugin is responsible for tracking byte order */
};
```

2.5 Plugins

GIMP uses a plug-in system which allows to develop new functionalities without having to modify the GIMP core. The loading, saving, and preview generation of PSD files is managed by a plugin called ‘file-psd’. We are now going to discuss the architecture of the plug-in architecture.

2.5.a Architecture

GIMP provides access to an API for manipulating and interacting with the images. This bridge API is called PDB (Gimp Procedure Database) [20] [21]. When GIMP starts up, it will look into a set of folders to find possible plug-ins, and add them to the procedure database. This allows to add plugins dynamically without having to compile GIMP again.
The plugin architecture requires the plugin code to declare some predefined functions. **init**, **quit**, **query**, and **run**.

**Query**: It is used by GIMP to detect if the binary file is a plug-in. It is called when GIMP starts up and is building the PDB.

**Init**: This function is called every time GIMP is started. This is called after GIMP has recognized the file as a plug-in.

**Quit**: This function is called when quitting GIMP.

**Run**: This function runs the plugin functionality.

The plug-in has only relevant code on the **Run** and **Query** functions.

2.5.b Debugging

Since GIMP executes plug-ins in a different threads, it is not possible to debug them directly. To allow debugging, libgimp looks for a special environment variable called `GIMP_PLUGIN_DEBUG`. If this variable is set, libgimp will halt the plug-in thread and print the process id. Then you can attach a debugger to the process id and debug it as you would do it with a normal program and continue the execution.

For example, to debug our PSD plugin, we will define the environment variable in the following way:

```
export GIMP_PLUGIN_DEBUG=file-psd,run
```

Then we can debug the plugin using with the following command.

```
gdb -p <process-id>
```
2.5.c Return values

A plugin returns different values depending on the execution result of the plugin:

- **GIMP_CALL_ERROR**: Gimp called the plug-in in the wrong way (missing arguments, incorrect arguments). This is an error in the way GIMP calls plugins.
- **GIMP_EXECUTION_ERROR**: There was an error in the execution of the plugin (example, the file was not found, the file is not a PSD file, etc).
- **GIMP_SUCCESS**: If there are no errors, the plugin returns success and GIMP assumes the operation finished successfully.

3 PSD Compatibility

3.1 Definition

For all the elements available in the PSD format, there will be some which are not compatible or directly translatable to GIMP. There will be other elements which are not translatable from GIMP into PSD elements. We will classify each feature in the following categories:

- **Compatible**: The resource can be fully imported, edited and exported back into a PSD file exactly in the same way. **Importing and exporting a compatible PSD file will result in exactly the same PSD file.**
- **Partially compatible**: These resources can be imported and edited in GIMP, but only partially or requires some special transformation to be able to do so. Also they are not exportable in the same way they were imported. **Importing and exporting a partially compatible PSD will result in different but equivalent PSD files.**
- **Not documented**: The documentation about the resource is incomplete or missing. **Those features will not be imported or exported and will be lost**. Reverse engineering PSD files to find out the undocumented parts of the format is out of the scope of this work.
- **Irrelevant**: Not useful or editable in GIMP. **Those will be imported into parasites and exported back identically.**
- **Not importable**: The resource cannot be loaded into gimp because it lacks the necessary functionality. **Those features will not be imported or exported.**
- **Not exportable**: The GIMP resource cannot be exported into PSD elements. **Those elements will not be added into the PSD image.**
Cases **Not fully documented**, **Not importable**, and **not exportable** will generate user warnings with the information that will be lost or the transformation that will be applied to the exported / imported image.

### 3.2 Format of the graphics

Since the binary specification of the PSD file can be tedious to read, we use graphical representations to make them more readable, here we present an example and explain how to interpret them. We might also omit some irrelevant fields for the sake of simplicity.

![Diagram](image)

*Illustration 2: Example graphic describing notation*

We have a PSD section, which contains 4 fields. The first character in each field describes its type. A number denotes the length in bytes of the field. An ‘S’ denotes a variable length string. Finally, ‘V’ defines a subsection of variable length, which can contain more fields and more subsections.

### 3.3 PSD Import

Although Adobe Photoshop is a private licensed software, the format it uses to save and store the files is free and published by Adobe [22]. The file format is binary and requires byte per byte parsing.

The PSD format has many versions and has been around for quite long. Because of lack of documentation for earlier versions, we will focus only in versions above 2. We will not consider support for version 2 or below.

This is a first preliminary comparison between both GIMP and Photoshop features based on the features described in both programs documentation. This will allow an early planning of the features to develop.

The basic structure consists of 5 sections, those are: **file header**, **color mode**, **image resources**, **layer and mask**, and **image data**. Note that each of those sections are divided in subsections of variable length which might contain more subsections in themselves.
Except for the file header, the rest of the sections have a variable length. The first field is always a 4 byte value containing the length of the rest of the section. This allows to jump whole sections to easily parse the document.

We will go through the PSD format specification with all it’s features and identify possible incompatibilities with gimp. Here we can the basic structure of each of the main sections. Lengths are defined in bytes.

3.3.a High level structure

![Illustration 3: PSD high level structure](image)

3.3.b File Header

The file header is the only section with a defined length. We describe the different fields in the following image:

![Illustration 4: File header fields](image)

The header describes the color mode of the file, the depth of the pixels, number of channels, and dimensions. The possible pixel depth values are:

- 1 bit  
  Compatible (conversion to 8 bit precision)
- 8 bits  
  Compatible
- 16 bits  
  Compatible
The different color mode values are:
- RGB   Compatible
- GRAYSCALE    Compatible
- INDEXED   Compatible
- CMYK    Compatible (conversion to RGB)
- LAB     Compatible (conversion to RGB)
- BITMAP Compatible (conversion to GRAYSCALE)
- DUOTONE Compatible (conversion to GRAYSCALE)
- MULTICHANNEL  Compatible (conversion to GRAYSCALE with extra channels)

The number of channels has a maximum of 56. Compatible (GIMP has no maximum).

3.3.c Color Mode Data
There is only a color mode section when the color mode in the file header color mode field section is set to INDEXED or DUOTONE. Otherwise the length of the section is set to zero and there is no data.

Illustration 5: Color mode data fields
For indexed color mode, the color mode contains the 256 colors that compose the indexed image palette of colors.

In case of duotone, the format is not documented, but the specification describes that the image can be imported and treated as a grayscale image, as long as the color mode data is exported back with the same data.

“Because duotones use different color inks to reproduce different gray levels, they are treated in Photoshop as single-channel, 8 bit, grayscale images. In Duotone mode, you do not have direct access to the individual image channels (as in RGB, CMYK, and Lab modes). Instead, you manipulate the channels through the curves in the Duotone Options dialog box.”[23]

- Indexed    Compatible
- Duotone   Compatible (Conversion to GRAYSCALE and add parasite)
3.3.d Image Resources

The resource data field depends on the type of Resource. We list now all the different resources available. For simplicity we omit the ID marked as obsolete in the specification.

**Resource types**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Compatibility with gimp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03E9</td>
<td>Macintosh print manager print info record</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03ED</td>
<td>ResolutionInfo structure.</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03EE</td>
<td>Alpha channels Names</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x03F0</td>
<td>Caption as a Pascal String</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x03F1</td>
<td>Border information</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>0x03F2</td>
<td>Background color</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>0x03F3</td>
<td>Print flags.</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x03F4</td>
<td>Grayscale and multichannel halftoning information</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03F5</td>
<td>Color halftoning information</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03F6</td>
<td>Duotone halftoning information</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03F7</td>
<td>Grayscale and multichannel transfer function</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03F8</td>
<td>Color transfer functions</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03F9</td>
<td>Duotone transfer functions</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03FA</td>
<td>Duotone image information</td>
<td>Not documented</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Compatibility</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0x03FB</td>
<td>Two bytes for the effective black and white values for the dot range</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x03FD</td>
<td>EPS options</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x03FE</td>
<td>Quick mask information</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0400</td>
<td>Layer state information</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0401</td>
<td>Working path</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0402</td>
<td>Layer group information</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0404</td>
<td>IPTC-NAA record.¹</td>
<td>Compatible [24]</td>
</tr>
<tr>
<td>0x0405</td>
<td>Image model for raw format files</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0406</td>
<td>JPEG quality.</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0408</td>
<td>Photoshop 4: Grid and guides information</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0409</td>
<td>Photoshop 4 only: thumbnail resource</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x040A</td>
<td>Photoshop 4: Copyright flag</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x040B</td>
<td>Photoshop 4: URL</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x040C</td>
<td>Photoshop 5: Thumbnail resource</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x040D</td>
<td>Photoshop 5: Global Angle effects layer</td>
<td>Not compatible</td>
</tr>
<tr>
<td>0x040F</td>
<td>Photoshop 5: ICC Profile²</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0410</td>
<td>Photoshop 5: Watermark</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>0x0411</td>
<td>Photoshop 5: ICC Untagged Profile</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0412</td>
<td>Photoshop 5: Effects visible</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x0413</td>
<td>Photoshop 5: Spot Halftone</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0414</td>
<td>Photoshop 5: Document ID seed number</td>
<td>Irrelevant</td>
</tr>
</tbody>
</table>

¹ IPTC-NAA is a standard for metadata information embedded in images. More specifically, this standard is published by the International Press Telecommunications Council (IPTC) together with the Newspaper Association of America (NAA). For more information see [1]

² An ICC profile, is a set of data that defines a color space. It is an standard format defined and maintained by the International Color Consortium. For more information see [2]
<table>
<thead>
<tr>
<th>Code</th>
<th>Feature Description</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0415</td>
<td>Photoshop 5: Unicode Alpha names</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0416</td>
<td>Photoshop 6: Indexed color table count</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0417</td>
<td>Photoshop 6: Transparency index</td>
<td>Not compatible</td>
</tr>
<tr>
<td>0x0419</td>
<td>Photoshop 6: Global Altitude</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x041A</td>
<td>Photoshop 6: Slices</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>0x041B</td>
<td>Photoshop 6: Workflow URL</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x041C</td>
<td>Photoshop 6: Jump to XPEP</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x041E</td>
<td>Photoshop 6: URL List</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0421</td>
<td>Photoshop 6: Version info</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0422</td>
<td>Photoshop 7: EXIF data 1</td>
<td>Compatible [24]</td>
</tr>
<tr>
<td>0x0423</td>
<td>Photoshop 7: EXIF data 3</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0424</td>
<td>Photoshop 7: XMP metadata</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0425</td>
<td>Photoshop 7: Caption digest</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0426</td>
<td>Photoshop 7: Print scale</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>0x0428</td>
<td>Photoshop CS: Pixel aspect ratio</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0429</td>
<td>Photoshop CS: Layer Comps</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x042A</td>
<td>Photoshop CS: Alternate Duotone colors</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x042B</td>
<td>Photoshop CS: Alternate Spot Colors</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x042D</td>
<td>Photoshop CS2: Layer selection ID</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x042E</td>
<td>Photoshop CS2: HDR Toning information</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x042F</td>
<td>Photoshop CS2: Print info</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0431</td>
<td>Photoshop CS3: Color samplers resource</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0432</td>
<td>Photoshop CS3: Measurement scale</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x0433</td>
<td>Photoshop CS3: Timeline information</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0434</td>
<td>Photoshop CS3: Sheet disclosure</td>
<td>Not documented</td>
</tr>
<tr>
<td>0x0435</td>
<td>Photoshop CS3: DisplayInfo structure</td>
<td>Not documented</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
<td>Compatibility</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0x0436</td>
<td>Photoshop CS3: Onion skins</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x0438</td>
<td>Photoshop CS4: Count information</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043A</td>
<td>Photoshop CS5: Print information</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043B</td>
<td>Photoshop CS5: Print style</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043C</td>
<td>Photoshop CS5: Macintosh NSPrintinfo</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043D</td>
<td>Photoshop CS5: WINDOWS devmode</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043E</td>
<td>Photoshop CS6: Auto Save File Path</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x043F</td>
<td>Photoshop CS6: Auto Save Format</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x0440</td>
<td>Photoshop CC: Path selection state</td>
<td>Compatible</td>
</tr>
<tr>
<td>0x07D0 - 0xBB6</td>
<td>Path information</td>
<td>Compatible</td>
</tr>
<tr>
<td>0xBB7</td>
<td>Name of clipping path</td>
<td>Compatible</td>
</tr>
<tr>
<td>0xBB8</td>
<td>Photoshop CC: Origin path info</td>
<td>Compatible</td>
</tr>
<tr>
<td>0xFA0 - 0x1387</td>
<td>Plug-in resources</td>
<td>Not compatible</td>
</tr>
<tr>
<td>0x1B58 - 0x1B5E</td>
<td>Image Ready data</td>
<td>Not compatible</td>
</tr>
<tr>
<td>0xF40</td>
<td>Photoshop CS3: Lightroom workflow</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>0x2710</td>
<td>Print flags information</td>
<td>Irrelevant</td>
</tr>
</tbody>
</table>

*Tabla 1: Image resources IDs compatibility*

3.3.e
3.3.f Layer and Mask information

This section contains all the information about the different layers, as well as the filters, effects, or masks attached to them. It has many sections and subsections.

Illustration 6: Layer and mask section

The Layer info section is divided in the following subsections:

Layer info

Illustration 7: Layers and Masks Information

Layer records

Illustration 8: Layer information subsection, Layer records

The clipping masks are compatible (imported as a normal mask, parasite added for exporting again as clipping mask) [25].

The list of the available Blend mode keys and it's compatibility with GIMP layer blend modes is the following. Keep in mind that some blend modes are exclusive to GIMP and therefore cannot be exported to PSD. When possible, some transformation will be made to be able to import an image with unsupported blend modes in gimp.
<table>
<thead>
<tr>
<th>Layer mode</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass through</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Normal</td>
<td>Compatible</td>
</tr>
<tr>
<td>Dissolve</td>
<td>Compatible</td>
</tr>
<tr>
<td>Dark</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Multiply</td>
<td>Compatible</td>
</tr>
<tr>
<td>Color burn</td>
<td>Compatible</td>
</tr>
<tr>
<td>Linear burn</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Darker color</td>
<td>Compatible</td>
</tr>
<tr>
<td>Lighten</td>
<td>Compatible</td>
</tr>
<tr>
<td>Screen</td>
<td>Compatible</td>
</tr>
<tr>
<td>Color dodge</td>
<td>Compatible</td>
</tr>
<tr>
<td>Linear dodge</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Overlay</td>
<td>Compatible</td>
</tr>
<tr>
<td>Soft light</td>
<td>Compatible</td>
</tr>
<tr>
<td>Hard light</td>
<td>Compatible</td>
</tr>
<tr>
<td>Vivid light</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Pin light</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Hard mix</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Difference</td>
<td>Compatible</td>
</tr>
<tr>
<td>Exclusion</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Substract</td>
<td>Compatible</td>
</tr>
<tr>
<td>Divide</td>
<td>Compatible</td>
</tr>
<tr>
<td>Saturation</td>
<td>Compatible</td>
</tr>
<tr>
<td>Color</td>
<td>Compatible</td>
</tr>
<tr>
<td>Luminosity</td>
<td>Not compatible</td>
</tr>
</tbody>
</table>

Table 2: PSD layer blend modes compatibility

Layer mask data
The layer and mask parameter data is marked as variable length because it might or not might be present with different lengths according to the flags field value.

**Layer blending ranges data**

**Illustration 9: Layer records subsection, Layer mask data**

The layer and mask parameter data is marked as variable length because it might or not might be present with different lengths according to the flags field value.

**Layer blending ranges data**

**Illustration 10: Layer records subsection, Layer blending ranges**

**Channel Image Data Section**

**Illustration 11: Layer channel image data**

The different compression modes are: **Raw data, RLE, ZIP without prediction, ZIP with prediction**. If the data is compressed, it needs to be uncompressed before importing it to GIMP.

**Global layer mask info Section**

**Illustration 12: Global layer mask info**
### Additional layer information

**Illustration 13: Additional layer information**

Following are a list of all the different data available. Again, we will list each one and define the compatibility with GIMP.

<table>
<thead>
<tr>
<th>Name</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjustment layer</td>
<td>Compatible</td>
</tr>
<tr>
<td>Effects layer</td>
<td>Compatible</td>
</tr>
<tr>
<td>Type Tool info</td>
<td>Compatible</td>
</tr>
<tr>
<td>Unicode layer name</td>
<td>Compatible</td>
</tr>
<tr>
<td>Layer ID</td>
<td>Compatible</td>
</tr>
<tr>
<td>Object-based effects layer info</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Patterns</td>
<td>Compatible</td>
</tr>
<tr>
<td>Annotations</td>
<td>Compatible</td>
</tr>
<tr>
<td>Blend clipping elements</td>
<td>Compatible</td>
</tr>
<tr>
<td>Blend interior elements</td>
<td>Compatible</td>
</tr>
<tr>
<td>Knockout setting</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Protected setting</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>Sheet color setting</td>
<td>Compatible</td>
</tr>
<tr>
<td>Reference point</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>Gradient settings</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Section divider setting</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Channel blending restriction setting</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Solid color sheet setting</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Pattern fill setting</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Setting</td>
<td>Compatibility</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Gradient fill setting</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Vector mask setting</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Type tool object setting</td>
<td>Compatible</td>
</tr>
<tr>
<td>Foreign effect ID</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Layer name source setting</td>
<td>Compatible</td>
</tr>
<tr>
<td>Pattern data</td>
<td>Compatible</td>
</tr>
<tr>
<td>Metadata setting</td>
<td>Compatible</td>
</tr>
<tr>
<td>Layer version</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>Transparency shapes layer</td>
<td>Compatible</td>
</tr>
<tr>
<td>Layer mask as global mask</td>
<td>Compatible</td>
</tr>
<tr>
<td>Vector mask as global mask</td>
<td>Compatible</td>
</tr>
<tr>
<td>Brightness and contrast</td>
<td>Compatible</td>
</tr>
<tr>
<td>Channel Mixer</td>
<td>Compatible</td>
</tr>
<tr>
<td>Color Lookup</td>
<td>Compatible</td>
</tr>
<tr>
<td>Placed Layer</td>
<td>Compatible</td>
</tr>
<tr>
<td>Linked Layer</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Photo Filter</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Black White</td>
<td>Compatible</td>
</tr>
<tr>
<td>Content Generator Extra Data</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Text Engine Data</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Vibrance</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Unicode Path Name</td>
<td>Compatible</td>
</tr>
<tr>
<td>Animation Effects</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Filter Mask</td>
<td>Not fully compatible</td>
</tr>
<tr>
<td>Placed Layer Data</td>
<td>Compatible</td>
</tr>
<tr>
<td>Vector Stroke Data</td>
<td>Compatible</td>
</tr>
<tr>
<td>Vector Stroke Content Data</td>
<td>Compatible</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Using Aligned Rendering</td>
<td>Not documented</td>
</tr>
<tr>
<td>Vector Origination Data</td>
<td>Not documented</td>
</tr>
<tr>
<td>Pixel Source Data</td>
<td>Not documented</td>
</tr>
<tr>
<td>Artboard Data</td>
<td>Not documented</td>
</tr>
<tr>
<td>Smart Object Layer data</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Saving Merged Transparency</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>User Mask</td>
<td>Compatible</td>
</tr>
<tr>
<td>Exposure</td>
<td>Not compatible</td>
</tr>
<tr>
<td>Filter Effects</td>
<td>Not fully compatible</td>
</tr>
</tbody>
</table>

Tabla 3: Layer and mask additional resources

3.3.g Image Data Section
This section contains the pixels of the visible Photoshop image. This is used by the thumbnail method, but not by the import method.

Illustration 14: Image data section
Note that it has the same format that layer mask channel data section.

3.4 PSD export
We are going to list the capabilities that GIMP has that are not compatible or completely compatible with the PSD format. We are just going to go through the sections which present issues and describe them.

3.4.a File Header
- Images with more than 56 channels are not compatible with PSD.
- Images with dimensions higher than 30,000 x 30,000 are not compatible with PSD.
- GIMP supports pixel precision that are not supported in PSD (16 and 32 bit floating point). GIMP also supports linear light pixel encoding additionally to the gamma light encoding supported in PSD. Those will be converted and the user will be warned about precision loss.

3.4.b Layer and Mask information section
The blend modes existing in GIMP but not compatible with PSD are [26]:

- Grain Merge
- Grain Extract

4 Existing PSD plugin
The code for importing and exporting PSD files is split in different files with a different code structure. They were developed by different persons. The import code was developed in 2007 by John Marshall and the export code by ‘Monigotes’ in 2000. Note that the plugin code has already been ported to Gegl. [27]

4.1 Import code
The import code is separated into 2 different steps. First, the file is parsed and all the data is accommodated in a big data structure. Then, this data structure is translated into GIMP image.

A first visual approach in comparison with the PSD specification of the import plugin reveals the following issues:

- Many nested brackets and convoluted conditions.
- Repetitive code blocks with the same functionality.
- Parsing functions don’t always correlate with PSD file sections and subsections.

More specifically, for each of the sections the following issues are observed:

File header:
The file header is parsed and used correctly.

Color mode:
Color modes CMYK, Multichannel and LAB are not imported. Duotone is imported as grayscale with a parasite.

**Image Resources:**
This section is completely skipped and none of the resources are parsed or imported at all.

**Layer and mask:**
Layer Blending ranges information is not parsed.
Adjustment layer info seems to be wrongly parsed. Needs tests to verify.
Global Layer Mask info field is skipped.
Additional Layer information skipped.

**Image data:**
This section is not used neither parsed when importing a file.

### 4.2 Thumbnail
The thumbnail generation code reads the image data section and generates a preview of the PSD file. It doesn’t handle image data sections with any ZIP compression type.

### 4.3 Export code
The export code writes all the PSD file converting the GIMP data structures at the same time it writes the PSD file.

- The code has less repetitive code blocks, since there are specific functions for repetitive tasks as writing a 32 bit integer, writing a pascal string, etc.
- The code doesn’t correlate very well with the psd file specification document.
- Some comments are unclear and misleading while reading the code.

**File header:**
Export code only exports in 8 bit color depth.

**Color mode:**
Indexed and duotone color images are correctly exported.

**Image Resources:**
Only resources 0x03EE, 0x0408, 0x03ed, 0x0400 and 0x040f are written to the psd file.

**Layer and mask:**
Layer and mask section length is apparently set incorrectly. Needs testing to verify.
Layer info section length is apparently set incorrectly. Needs testing to verify.
Layer mask / adjustment layer data section is written empty.
Layer blending ranges is written empty.
In additional layer information, only layer name and sheet color setting is written.

**Image data:**
This section is written always in RLE compressed mode.

### 4.4 Open bugs regarding PSD import / export

For further determining the missing features we will treat the different sections by separate and test them with unitary tests. We will produce a set of test files with each section by separate. We will also look at the open bugs with keyword ‘psd’ in the GIMP bug tracking system. [28] On the 26/04/2017 there are 52 different bugs without resolution. Looking through the found bugs we filter the 52 different bugs into a list of 16 relevant ones for the plugin.

For each bug, we are going to identify if it is related to some bug in the code or a lack of implementation of a feature. Note that we eliminated the duplicated bug reports to avoid redundancy.

<table>
<thead>
<tr>
<th>Bug</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store channel colors when saving as PSD</td>
<td>Missing feature in export</td>
</tr>
<tr>
<td>Mask positions move to top-left when saving as PSD</td>
<td>Possible bug in export</td>
</tr>
<tr>
<td>Support importing CMYK files</td>
<td>Missing feature in import</td>
</tr>
<tr>
<td>PSD stroke not importing</td>
<td>Missing feature in import</td>
</tr>
<tr>
<td>Add support for Photoshop styles and adjustment layers</td>
<td>Missing feature in import</td>
</tr>
<tr>
<td>PSD plugin should support path data (missing export)</td>
<td>Missing feature in export</td>
</tr>
<tr>
<td>‘file-load-psd()’ crashes importing file (file provided)</td>
<td>Possible bug in import</td>
</tr>
<tr>
<td>Opening PSD files doesn’t apply the correct color profile</td>
<td>Missing feature in import</td>
</tr>
<tr>
<td>Issue</td>
<td>Note</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Text information is lost when doing PSD export</td>
<td>Missing feature in export</td>
</tr>
<tr>
<td>File format support test suites: PSD via ‘psd-tools’</td>
<td>Feature request</td>
</tr>
<tr>
<td>Incorrect import PSD file format (transparency related)</td>
<td>Possible bug in import</td>
</tr>
<tr>
<td>PSD export and import miss the undo history</td>
<td>Missing feature in import / export</td>
</tr>
<tr>
<td>Loosing layer groups when export to PSD</td>
<td>Possible error in export</td>
</tr>
<tr>
<td>Support layer masks on layer groups</td>
<td>Possible error in import</td>
</tr>
<tr>
<td>Colored layer support - data loss on import from Adobe Photoshop</td>
<td>Possible error in layer import</td>
</tr>
<tr>
<td>Mask positions may be incorrect when opening a PSD</td>
<td>Possible error on layer import</td>
</tr>
<tr>
<td>PSD import: add warning for missing capabilities</td>
<td>Missing feature</td>
</tr>
<tr>
<td>‘file-load-psd.exe’ crashes importing file</td>
<td>Bug</td>
</tr>
<tr>
<td>File format support test suites: PSD via ‘psd-tools’</td>
<td>Missing feature</td>
</tr>
<tr>
<td>Incorrect import PSD file format</td>
<td>Possible error on layer transparency import</td>
</tr>
<tr>
<td>Cutting off layer mask when it’s outside layer boundary while opening PSD file in GIMP</td>
<td>Possible bug</td>
</tr>
</tbody>
</table>

*Tabla 4: List of open bugs regarding PSD import / export*
5 Planning

5.1 Approach
The existing plugin code is highly difficult to maintain and it is very difficult to add modifications to it. For instance, the file that manages the loading of PSD files contains over 2300 lines of code itself [29]. The code is not modular and it is not divided properly in functions that read each section.

On the other hand, there is already a basic working code which manages some PSD files and loads layers and mask sections. We can make use of the existing code as a reference point and testing in the first steps of development.

We want to design a new plugin which is modular and allows to execute unit tests on the different sections and subsections. In this way, we can add more features or maintain the source code more easily. When developing, we will honor the GNU coding standard guidelines [30].

We define a basic minimal PSD file, and all its variants. Then we are going to develop the code to support this minimal file and write tests for those files. From there on, with a stable basic code, we will add more features gradually using a test driven development. We will write the tests about how the code should behave and code the features so the tests pass.

5.2 The Minimal PSD file
We talk about the simplest PSD file to a PSD file which contains the least amount of features and is a valid PSD file. This PSD file will serve as a starting point for developing the first code base. Once we can import and export minimal PSD files we will add tests for those files and continue the development of more features one by one.

<table>
<thead>
<tr>
<th>File header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty colormode section</td>
</tr>
<tr>
<td>Empty image resources</td>
</tr>
<tr>
<td>Empty layers and mask</td>
</tr>
<tr>
<td>Merged image section</td>
</tr>
</tbody>
</table>

\[Drawing\] 1: Minimal PSD file structure
The minimal PSD file has a RGB color mode (of any allowed depth), with or without transparency channel (no additional alpha channels), with no image resources, only one layer (stored in the merged image section), and the merged image data stored without compression.

5.3 Main Data structures

We define two main data structures for handling PSD to GIMP export, and GIMP to PSD to PSD export. Those structures contains data that needs to be available during the reading / parsing of most of the sections / subsections. There are defined as follows:

```c
typedef struct{
    gint16 color_mode;
    gint16 depth;
    gint32 width;
    gint32 height;
    gint32 channels;
    Babl* format;
} PSDimage;

typedef struct{
    GimpBaseType image_base;
    GimpPrecision precision;
    GimpImageType image_type;
    Babl* format;
} GIMPimage;
```

When can also create specialized functions in transforming one data structure to the other as follows. Note that the gimp to PSD transformation allows to export the image in a different color mode than the supported by GIMP. We elaborate more on this in the next section “Export Graphical Interface” below.

```c
psd_to_gimp_structure();
gimp_to_psd_structure( gint16 export_colormode, gint16 export_pixeldepth);
```

5.4 Code structure

We are going to structure the code using the same sections as defined in the PSD format. We will also divide the code in multiple files, each one managing a different section of the format. Managing the ‘#include’ declarations in the C code we can insulate and make modular the code and the variable scopes. This is a rudimentary way of simulating independent ‘objects’ in C language. We will also create reusable functions for reading and writing operations which are used very often (read32bitInteger, write32bitInteger, readPascalString, etc).
5.5 Exception management

We differentiate between **errors** (a situation not recoverable and causes the import to fail) from **warnings** (some feature cannot be imported, but the import procedure continues) [31].

5.5.a Errors

We use a Glib structure called GError. When an error happens in a function returns ‘-1’ and fills the GError variable with the message error that made it happen. This error message is embedded into the plugin return parameters and displayed as a popup window in GIMP.

Each function checks the return every time it calls another function, if the result is ‘-1’, the result is passed back to the parent function call.

5.5.b Warnings

A warning will present the user with a pop up window informing with the situation. Since there can be multiple warning messages, we will keep a warnings message to keep all of them and display a list in the end of the executing (unless there is an error, then only the error will be shown).

5.6 Export User Interface

There are many color modes that are converted into one of the three supported color modes in GIMP (RGB, GRAYSCALE, and INDEXED). Usually, we would store the original color mode in a parasite and then export it in the same way it was imported.

In most of the cases this black box model about how features get transferred from PSD to GIMP and vice versa works well. In this case, it doesn’t. It would be more clear and transparent to the user to present a GUI window with the color mode in which the file is going to be exported to. This allows users to export images intentionally in color modes different than the original and not supported in GIMP (the most prevalent case might be CMYK) [32].

We present here some figures about how the actual export window looks like.
The code for creating the Graphical interface was made taking this [33] code as a reference from the JPEG plugin export dialog.

5.6.a Update logic
Here we describe the possible available conversions depending on the original color mode of the GIMP image.

<table>
<thead>
<tr>
<th>Gimp color mode</th>
<th>Available output color modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexed</td>
<td>Bitmap, Indexed, Grayscale, RGB, CMYK, Lab</td>
</tr>
<tr>
<td>Grayscale</td>
<td>Grayscale, Bitmap</td>
</tr>
<tr>
<td>RGB</td>
<td>Bitmap, Indexed, Grayscale, RGB, CMYK, Lab</td>
</tr>
</tbody>
</table>

**Tabla 5: Available color modes from GIMP images**

For each of the available color modes there is a set of available pixel depths:

<table>
<thead>
<tr>
<th>Color mode</th>
<th>Possible PSD pixel depths</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITMAP</td>
<td>1 bit</td>
</tr>
<tr>
<td>INDEXED</td>
<td>8 bits</td>
</tr>
<tr>
<td>GRAYSCALE</td>
<td>8, 16, 32 bits</td>
</tr>
<tr>
<td>RGB</td>
<td>8, 16, 32 bits</td>
</tr>
<tr>
<td>CMYK</td>
<td>8, 16 bits</td>
</tr>
<tr>
<td>LAB</td>
<td>8, 16 bits</td>
</tr>
</tbody>
</table>

**Tabla 6: Available pixel depths per each color mode**
Every time a color mode is selected, we need to update the pixel depth menu below to only display the available pixel depths in that color mode.

This is done using GTK signal system. We attach a “changed” signal that triggers a function every time the color mode selection mode is changed.

5.6.b Default values
By default, the selected values will be the ones matching the original image color modes and pixel depth. In case of unsupported GIMP pixel depths, the following conversions will be applied. Note there will be a loss in precision in those exported images.

<table>
<thead>
<tr>
<th>Original pixel depth</th>
<th>Converted pixel depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bit floating point</td>
<td>32 bits</td>
</tr>
<tr>
<td>32 bit floating point</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

Tabla 7: Higher pixel depth transformations

5.7 Testing framework
We are going to differentiate between two types of tests. Unit tests and integration tests [34].

**Unit tests** are executed in the plugin source code. They check the individual functions and sections of the plugin. The testing code will remain in the plugin source folder and executed directly. Since the code relies on reading and writing file sections, we can write unit tests that check the export and import code together. We here describe the basic test structure with pseudo code:

```plaintext
File := create_empty_file()
Input := Input values for the section
Output := Output values of reading the section
write_test_section(File, Input); //Outputs the section we are testing in the file
rewindFile(); //rewinds the file cursor to the beginning
Output := read_test_section(File); //reads the previously
//Check that input and output are equal.
```

**Integration tests** checks how the plugin integrates as a whole with GIMP. It simulates a GIMP environment with its configuration files and emulates the behavior GIMP would have.
GIMP provides an undocumented infrastructure for performing integration tests [33], but some work is required to make it work with plug-ins. The integration test will have the following structure:

1. Create a specific image programmatically in GIMP
2. Export the image into a PSD file calling the plugin through GIMP
3. Import the image again
4. Check the resources are still there and have the right data

5.8 File parsing strategies

The parsing of binary files involves that reading a single byte more or a single bit less will cause the rest of the code to fail. When possible, we will use the following strategies when parsing the file:

5.8.a Length defined sections

When we have a section with a length field defined, we run the code that parses that section, but later, we set the file pointer to the end of the section. If there is a bug or error that makes the subsection function read more or less bytes than there are in the section, the rest of the code will not fail. This also introduces some tolerance to files with incorrect data without failing the whole import. We describe the procedure through pseudo code:

```
Length := read section length from file
Position := file cursor position
End := position + length

Call parsesectionfunction();

Set cursor position ();
```

5.8.b Export section lengths

In the same manner, but inverse, we set the same strategy for writing sections and creating correct values for its lengths. We run store the position of the section length field. Then, we write a 0 temporally and execute the subsection export function. When the call has finished, we get the file cursor position and calculate the length. Go back to the section length position and write the written length. Finally we set the cursor back to the original position.
5.8.c Character signatures
In some sections of the file, there are 4 character signatures that indicate the beginning or ending of a section. For example, in the beginning of the file, the signature “8BMS” is written to indicate that the file is a PSD file.

Not finding a signature in the expected place is another way to check if there was a overread of bytes and a potential bug. In this case the strategy will be to make the plugin fail and stop reading.

5.8.d Reading Pseudo code
We now determine a general pseudo code for parsing a PSD section is defined.

```
Open file

read_header()
If (color_mode IS Indexed OR Duotone)
    read_colormode();

    // Image resources section
    length = readLength();
    end = current position + length;
    ReadImageResource();
    setPosition(End);

    //Layer and mask section
    Length = readLength();
    End = current position + length;
    ReadLayers();
    setPosition(End);

    //Merged Image Data
    readMergedImage();

    If NOT end of file
        ERROR
```

The Layer and Mask mode section doesn’t define any signature in the beginning of the layer or the length of each layer. Therefore none of the techniques described can be used to read some layers and some not. Either the whole layer and mask section is right or we will not read any layer.

5.9 Repository management
The code is managed in a Github repository, we will fork the git repository into our branch, and when we are done developing we will merge it back to the original branch.
To do so, we clone the original gimp repository and add another remote repository pointing to Github. Then we can add changes, commits, and push changes to the Github account. Once all the work is completed, the code can be merged again with the main repository.

The code is hosted in the following location: https://github.com/GuiuRocafort/gimp-fork-plug-in-psd-improvement

5.10 Tasks
We define the tasks to develop chosen approach:

- Setting of the development environment
- Build an empty functional plug-in
- Create code structure with empty files and functions
- Prepare code for adding unit tests
- Develop support for the minimal PSD file
- Develop support for compression modes
- Develop export GUI
- Add rest of color modes (with import/export transformations)
- Add support for layer and masks section
- Add support for image resources section
- Create integration tests

5.11 Temporal planning
We not define a temporal planning for the tasks described above. Note that the steps are linear and need to be executed one after the other because each one depends on the previous one. Therefore there is no need for planning the tasks using a Gantt diagram.

There is a relatively high uncertainty in the tasks, because it requires programming, and dealing with several libraries and other API. GIMP is a complex software and makes uses many libraries and we can encounter bugs in them, missing features, or just that they are difficult to use.
| **Setting of the development environment** | March, 21st |
| Build empty functional plug-in            |            |
| Create empty code structure               |            |
| Prepare code for unit tests               |            |
| Develop support for minimal PSD           |            |
| Develop support for compression modes      |            |
| Export GUI                                |            |
| Add support for the rest of color modes   |            |
| Layers and mask section                   |            |
| Support image resources                   |            |
| Create integration tests                  | June, 10th |

*Drawing 2: Tasks temporal planning*
6 Execution

In this chapter, we will describe the relevant steps or processes developed during the execution of the tasks defined in the previous chapter.

6.1 Setting up the development environment

The development environment for programming and developing the software is going to be a Virtualbox virtual machine with Debian 8.8. The GUI installed will be Xfce to reduce the size of the virtual machine.

Once we have the operating system, we proceed installing the development tools we are going to use. Git and Emacs.

We also install the necessary packages for compilation and development

```
apt-get install autogen autoconf2.13 build-essential libtool intltool
libjson-c-dev libjson-glib-dev
apt-get build-dep gegl babl gimp
```

We need to compile some libraries manually because the development version of Gimp needs the newest versions from those libraries.[35] The versions in the repository are too old. We will be compiling the libraries and installing them in a local folder. We need to define some environment variables first:

```
mkdir $HOME/gimp-install/
```

Then, we need to modify some environment variables so Debian can find the new compiled binaries. Those lines can be added to the file ‘.bashrc’ so they are defined automatically every time you execute a console.

```
export INSTALL_PREFIX=$HOME/gimp-install/
export PATH=$INSTALL_PREFIX/bin:$PATH
export PKG_CONFIG_PATH=$INSTALL_PREFIX/lib/pkgconfig:$PKG_CONFIG_PATH
export LD_LIBRARY_PATH=$INSTALL_PREFIX/lib:$LD_LIBRARY_PATH
export G_DEBUG_MESSAGES=file-psd,run
```

We fetch the latest releases of the libraries. Babl, Gegl, gexiv2, and libmypaint.

```
git clone git://git.gnome.org/babl
```

```
git clone git://git.gnome.org/gegl
```

```
wget https://download.gnome.org/sources/gexiv2/0.10/gexiv2-0.10.6.tar.xz
```

```
wget https://github.com/mypaint/libmypaint/releases/download/v1.3.0/libmypaint-1.3.0.tar.xz
```

For each the libraries we are going to execute the next commands:

```
./autogen.sh
./configure --prefix=$INSTALL_PREFIX --enable-debug --enable-profiling
make -j4
make install
```

45
Once we have compiled and installed all the libraries in the local folder, we can go ahead and get the latest gimp code from the repository. Then we are going to compile it disabling some features (python support, and GTK documentation generation). This makes it easier to compile and requires less libraries. Since psd-import and psd-export code doesn’t use python, it doesn’t affect the purpose of this work.

```
git clone git://git.gnome.org/gimp
./configure --prefix=$INSTALL_PREFIX --disable-python --disable-gtk-doc
--enable-debug --enable-profiling
make -j4
make install
```

### 6.2 Build empty functional plugin

For this task, we create an empty plugin, with the query and run functions. In the query function we register the procedures for loading, saving, and loading a preview of the image (thumbnail). This is done through the following PDB calls:

```c
/* File load */
gimp_install_procedure (LOAD_PROC,
    "Loads images from the Photoshop PSD file format",
    "This plug-in loads images in Adobe "
    "Photoshop (TM) native PSD format.",
    "John Marshall",
    "John Marshall",
    "2007",
    N_("Photoshop image"),
    NULL,
    GIMP_PLUGIN,
    G_N_ELEMENTS (load_args),
    G_N_ELEMENTS (load_return_vals),
    load_args, load_return_vals);

gimp_register_file_handler_mime (LOAD_PROC, "image/x-psd);
gimp_register_magic_load_handler (LOAD_PROC,
    "psd",
    ",
    "0,string,8BPS");

/* Thumbnail load */
gimp_install_procedure (LOAD_THUMB_PROC,
    "Loads thumbnails from the Photoshop PSD file format",
    "This plug-in loads thumbnail images from Adobe "
    "Photoshop (TM) native PSD format files.",
    "John Marshall",
    "John Marshall",
    "2007",
    NULL,
    GIMP_PLUGIN,
    G_N_ELEMENTS (load_args),
    G_N_ELEMENTS (load_return_vals),
    load_args, load_return_vals);
```
gimp_register_thumbnail_loader (LOAD_PROC, LOAD_THUMBPROC);

gimp_install_procedure (SAVE_PROC,
    "saves files in the Photoshop(tm) PSD file format",
    "This filter saves files of Adobe Photoshop(tm) native PSD format. These files may be of any image type supported by GIMP, with or without layers, layer masks, aux channels and guides.",
    "Monigotes",
    "Monigotes",
    "2000",
    N_("Photoshop image"),
    "RGB", GRAY*, INDEXED",
    GIMP_PLUGIN,
    G_N_ELEMENTS (save_args), 0,
    save_args, NULL);

    gimp_register_file_handler_mime (SAVE_PROC, "image/x-psd" );
gimp_register_save_handler (SAVE_PROC, "psd", "") ;

Then, we write a run function which checks the input parameters and returns GIMP_CALL_ERROR when necessary to sanitize the input variables.

Once this is done, we check the plugin is correctly introduced in the procedure database.

### 6.3 Create empty code structure

We create all the code files and definitions with empty functions. We create a different C file for each section / subsection of the PSD format. We define the input and output values and we set the code for error management and the return values.

The structure of the files is as follows ( only the h files are listed ):

```
psd.h
common.h
psd-save.h
psd-load.h
psd-thumb.h

save/psd-save-header.h
save/psd-save-colormode.h
save/psd-save-img-res.h
save/psd-save-layermask.h
```
6.4 Prepare code for unit tests

For running unit tests, we make use of Automake test suite functionality [36]. We are going to place the unit tests in a subfolder ‘tests’. The tests need to compile with code from the psd plug-in. This is made using a custom Makefile.am which tells Automake how to build the tests [37]. We also need to add “plug-ins/file-psd/tests/Makefile” to the file ‘configure.ac’.

Once the changes are done we navigate to the root folder and execute ‘autoreconf’ to force Automake to rebuild all the Makefile files. Then the tests can be executed by running “make check”. If we run the plugin folder ‘run check’, it will execute all the tests.

6.5 Babl bug fixed

During the development of the tests, I encountered a bug in Babl. A bug file was opened, and I proposed a patch for fixing the issue. The patch got accepted and now is in Babl master repository [38].

6.6 Develop support for the minimal PSD file

Support for the minimal PSD file includes the parsing and writing of PSD header section, and the parsing / writing of the image data section. In the rest of the sections, code for skipping the whole section was introduced.

The main difficulty of this part was understanding how Gegl buffers work, and learning about color modes, and pixel depths. Finally, a general code for loading raw image data was developed. Code for future addition of compressed modes was also included. The strategy is to uncompressed the data and then called the method developed for raw data.
6.7 Add support for rest of color modes

Adding reading support for the rest of color modes was relatively easy, Babl manages the pixel type conversions. Exporting the data was more challenging due to the need of a Graphical User Interface. The compatibility logic for each color mode and allowed pixel depth needed to implemented.

6.8 Add integration tests

The integration tests could not be implemented due to missing features in GIMP integration test environment. The environment does not have support for testing plugins and it is intended only for testing core features. The lack of documentation in the test infrastructure made the task even more difficult.

There is also a compilation bug related to the integration tests which couldn’t be resolved to the date [39].

7 Results

7.1 Achievements

We successfully realized a detailed compatibility analysis between GIMP and the PSD format. Based on that we successfully planned and designed a new plugin to export and import PSD files to GIMP. Finally, we rewritten the plugin code and implemented some of the PSD features.

The resulting code result of this work is available in the Github repository [40].

The code is a robust and well structured code, which doesn’t support all PSD features, but provides a good restructuring and shows great promise for future expansion. The main advantages that this code provides over the old plugin are:

- Robust and modular parsing infrastructure for PSD files with partial bug / file error handling.
- Clear error and warning management protocols.
- Separated functions for common operations, thus making much easier to add new code, and reducing considerable the number of lines of code.
- Added Graphical User Interface for choosing export color mode.

The implemented features in the current code are:

- Import of all color modes and depths (except multichannel).
• Export of all color modes and depths (except multichannel).
• Export Dialog for exporting different color mode / depth.
• Layers and masks section basic import and / export. (no layer resources)
• Proper reporting of errors / warnings.
• Unit tests.

Since the code doesn’t support the same features than the old code supports, it is not ready yet for merging with the GIMP master repository. We hope that this will be possible in the future.

7.2 Future work

The code produced can be expanded to support more PSD features for importing and exporting. The following features were not implemented and would be useful to implement in the future:

• Image resources
• Layer and masks resources
• Better algorithm for compression output decision

Another point to develop would be to fix the integrity testing system. Also, it needs to be documented and expanded to include integration tests involving plugins and not just the core functions. At this point, the bug requires investigation and it is not yet fully defined and reproducible.

Once the rest of the PSD features are added, the unit tests for those features created, and the integrity tests fixed, the code can be proposed for revision and eventually merged into GIMP main repository.

It could be possible to use Photoshop plugins code [41]. This would allow to use the plug-in information stored in the PSD files. Although this would require some important development in the GIMP core, and libraries. There is also the issue with code licenses, since most Photoshop plugins would not compatible with GIMP GPL.

7.3 Conclusions

Developing a GIMP plugin was a very challenging task which required many more skills and effort that I initially thought in the beginning. I had to get familiar with many library API like Glib, GTK, GNU Coding standards, Gegl, Babl, and GIMP itself. This familiarization with the API take a lot of time and effort.
This effort allowed me to appreciate the work of all people who support and maintain free software systems. A project like GIMP wouldn’t be possible without the support of people maintaining Glib, GTK, GNU Tools, the Linux kernel itself, GNU build tools, and a very long list of etceteras.

The experience also showed me to be more critical when thinking about code and programming, to value more the importance of software architecture through reading other people code.
8 Glossary of terms

**GIMP**: Gnu Image Manipulation System Image editing program

**PSD**: Photoshop Document file format

**CMYK**: Color mode used in printing which consists in 4 colors (Cyan Magenta Yellow and Black).

**LAB**: Color mode based on Light and two components a, b. Based on human perception of colors.

**IRC**: Internet relay chat. Provides chat communication in real time

**Glib**: C low level library. Extends standard C functionalities adding making programs more portable and easier to code.

**GTK**: (GIMP Toolkit). Library for creating and managing graphical user interfaces.

**Automake**: Build system by GNU, it allows to compile complex programs more easily

**Pixel depth**: Number of bits each pixel takes in a channel.
9 Bibliographic references


