BSc-project IN3405

Albumprinter file format

Find an open file format to replace the current proprietary format

Students
1169327 An Ngo
1048031 Xinan Zhu

August 6, 2008

Supervisors
Klaas Waslander
Sander Nagtegaal

Albumprinter B.V., Amsterdam

Supervisors
ir. Charles van der Mast
ir. Matthijs Sepers

The Faculty of EEMCS
Delft University of Technology

I. PREFACE

In order to complete the Bachelor's degree, students at the Delft University of Technology majoring in Computer Science need to do a BSc. Project. This project means for the students that they will perform an external internship in a group. The goal of this project is for the student to gain practical experience working in an actual development environment outside the university.

This thesis paper is written as result of such a BSc. Project carried out by An Ngo and Xinan Zhu at courtesy of Albumprinter Solutions, Amsterdam. This has been a project of three months starting from the 7th of April till the 7th of July, that seemed to have flown by.

During this period the "colleagues" at Albumprinter Solutions has been nothing but helpful, our thanks goes out to all of them. Especially we would like to express our gratitude to the X-editor team, without whose help we wouldn't be able to do anything at all.

Furthermore we need to mention our supervisor Klaas Waslander and Sander Nagtegaal who have been key in starting this project and providing us with their help and time.

An Ngo and Xinan Zhu

II. SUMMARY

At Albumprinter Solutions an internal file format is used at in the current process of creating a photo album. There are several issues with the fact that it's a self developed proprietary format:

- Maintenance of the format
- No third party development possible
- No printing industry support

The decision is made to address this problem in a project, alongside the improvement of the newly developed X-editor for consumers. There are several candidate file formats on the horizon, such as Adobe Mars-project, PPML, Microsoft XPS etc.

Research into the different file formats has lead to the decision to do a prototype with MS XPS. XPS (XML Paper Specification) is a XML based zip container describing the content and layout of a document, as was Adobe Mars. However due to uncertainties in sustaining and further development of Adobe Mars and the printing industry's support for XPS, the choice has been made favorable toward XPS.

A prototype is developed in the Adobe Flex® for demonstrating the possibility of creating a photo book in the X-editor and formatting it to XPS for printing. First of all a XPS-AP (Albumprinter XPS) is designed to be used to represent the photo album. Secondly the prototype application is designed and implemented.

III. TABLE OF CONTENTS

1 INTRODUCTION

Due to the growth that Albumprinter Solutions have been made in the last years and the growth that it foresees for the near future it has been looking at a replacement for their in-house standard of for the representation of their product.

There are several benefits to this action:

- By replacing the private proprietary standard by an open standard file format Albumprinter hopes to attract 3rd party developers as well as getting the in-house developers a more easy start.
- A cost benefit occurs when using an open file format, there is less need for maintenance on the format. E.g. readers and documentation.
- By choosing a new standard backed by the printing industry it could be that processes can be simplified and leading to a more efficient work process.

These benefits have triggered the start of a project to find and prototype a new file format that could possibly replace the current proprietary standard. At the same time as a new editor called the X-editor has been recently developed, this is forms a window of opportunity to try out a new format for storage of the album information within the application.

This paper discusses this process of prototyping a new file format for Albumprinter. First of all this paper will start out by explaining what kind of choices need to be made on deciding the new file format. It continues by highlighting the possible candidates and the final choice for the proof of concept. The final chapters will show what has been done in order to show the proof of concept and the paper closes with issues still left open at the end of the project and future recommendations on whether to use the chosen format or not.

Models and other examples can be found in the appendices.

1.1 PROBLEM STATEMENT

From the following task: “Find an open file format to replace the current proprietary format used within Albumprinter”, we derived to the following problem statement:

“Is there an open file format that can replace the current proprietary format without loss of quality of the product and extensibility for expansion?”

This paper will be using this problem statement as its core and form the leading thread.

From the problem statement there are several goals to be extracted and redefined into tasks.

Main goal: Finding a replacement file format.

Win conditions:

- Open standard
- Implementable in the X – editor
- Provide better flow: Digital photo album → Physical photo album
- New format contains enough information to print the album as represented in the editor

Complimentary conditions:

- Able to open in the X - editor
- New format is expandable for new features in the editor
- Compatibility with MS-Office
- Allows custom fonts
- Original picture can be kept for reuse

The current situation:

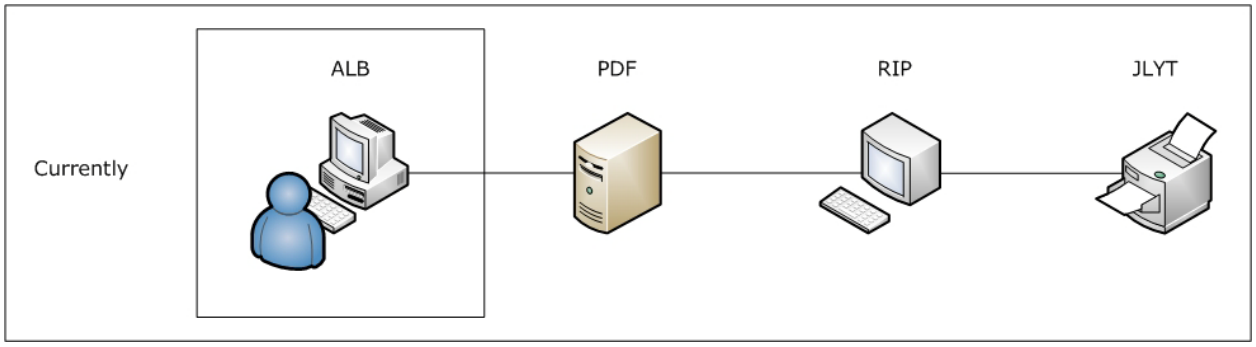


Figure 1: Current situation

The current situation of the file format transitions at Albumprinter is shown in figure 1. The proprietary file used at Albumprinter is called .ALB, and is used at the application level of the Albumprinter software to store album properties. Afterwards the ALB is converted to a PDF, and this PDF is sent to the RIP station to prepare the file for printing. At the printing stage the file is a JLYT, a file which is readable by printers. The main goal of this project is to find a format to replace the ALB in the first step by a more open format. The ideal situation would be to find a format that also replace the PDF in the second step as well.

1.2 APPROACH

The approach of the project is described in the Plan of Action, which can be found in Appendix C2. In the Plan of Action the approach is described in detail and also included a timetable of the project.

2 RESEARCH

The research into finding a possible solution to the problem consists of 4 parts, first of all by using the win and complimentary conditions a list is created that contains the features of the candidate file formats must support [2.1]. Secondly a look at the current printing market and software industry provides us with a list of possible candidate file formats [2.2]. Thirdly a comparison table is created from step one and two [2.3]. Finally we combine step one, two and three to choose a candidate for prototyping. This chapter describes the research process step by step before a design for the new file format and its prototype software can be constructed.

2.1 IMPORTANT FILE FORMAT ASPECTS

The first step in the research phase is to create a list of aspects that the candidates file formats must support. There are different types of aspects. A distinction is made between main aspects and secondary aspects.

2.1.1 MAIN ASPECTS

- **Document fidelity (WYSIWYG)**
  "What You See Is What You Get"; what you see on the screen, will be exactly printed on paper. This is clearly a crucial aspect, as Albumprinter provides services to create photo books on the computer, which will be printed to physical photo books. The final product has to be same as the product which is made with the photo book software application.
- **Open specification**
  The format should be an open format. This will support the usage of 3rd party software and it would provide less maintenance of the own proprietary format.
- **Original resources can be retrieved**
  Data that are stored in the format should be retrievable for reuse and editing. For Albumprinter it has advantages to retrieve original resources as images and text for editing at a later stage.
- **Portable / Multiplatform**
  The format should be portable and multiplatform. It should support the usage on different platforms. Albumprinter provides applications which are multiplatform.
- **Sustainability**
  The support by the organization of the format is guaranteed for the coming years.
- **Support by the printing industry**
  The printing industry should support the format.

2.1.2 SECONDARY ASPECTS

- **PDF conversion**
  In the transition stage it is important to convert the new file format to the PDF.
- **License agreement (Royalty-free)**
  Free usage of the format.

2.2 FILE FORMATS CANDIDATES

As a second step several possible file format candidates of the printing markets and software industry are selected for further research whether they are a realistic candidate for the prototype. After a research the following formats are possible candidates for prototyping as a new Albumprinter file format:

- **Microsoft XPS** (Microsoft, Microsoft XPS, 2003)

An open fixed-document format based on XML. It has similarities with PDF and PostScript.

- **Adobe Mars** (Adobe, 2006)
  Mars is an open XML-based format as a next generation of PDF.
- **PODi PPML** (PODi, 2008)
  PPML is an XML-based industry standard printer language.
- **Sun/OASIS ODF** (OASIS, 2003)
  An XML-base extensible language for office documents. ODF is the format of the OpenOffice package.
- **Microsoft OOXML** (Ecma, 2006)
  An XML-base extensible language for office documents. OOXML is the format in the Office 2007 package of Microsoft.

2.3 COMPARISON TABLE

For a comparison of the candidates a comparison table is created with the information collected during step one and two. The comparison table gives a better overview and will be useful for the next step in the research phase, in where the file format is selected as the prototype for the new Albumprinter format.

Table 1 shows only a section of the comparison table containing the main aspects. For a complete comparison table, see Appendix C4.

General File Format	Mars	XPS	PPML	ODF	OOXML
Ensure document fidelity (WYSIWYG)	●	●	●		
Open specification	●	●	●	●	●
Original resources can be retrieved	●	●		●	●
Portable / Multiplatform	●	●		●	●
Sustainability	□	●	●	●	●
Supported by printing industry	?	●	●		
PDF conversion	●	●	●	●	●
License agreement (Royalty-free)	?	●	●	●	●

● = supported, □ = partly supported, ? = unknown

Table 1: Comparison table

2.4 SELECTING PROTOTYPE FORMAT

In the last stage of the research a consideration has to made which format is the best candidate for prototype as the new Albumprinter format. The format for the prototype will be extracted from the previous steps and especially from step three, the comparison table.

As seen in the comparison table, not all formats support the main aspects. It is clear that ODF and OOXML can be turn down as candidates. Both formats doesn't support the WYSIWYG aspect, whilst this is necessary for the Albumprinter format. ODF and OOXML are also not supported by the printing industry.

As we look at PPML, it is clear that one aspect is not supported. PPML is not a portable format. Hereby PPML can also removed from the list.

There are two formats remaining; XPS and Mars. Both are comparable and look promising to use as the new Albumprinter format. XPS and Mars are XML-based document format and are similar to PDF. MARS was explicitly designed to be an XML representation of PDF itself. XPS was designed with the same goal in mind as the original PDF standard - a printable format- and as such a direct competitor for PDF and MARS. Albumprinter uses PDF as a format to feed the RIP stations for printing. The advantage of XPS and Mars over PDF is that they are both an open format with a better structural file hierarchy, which means that XPS and Mars are formats that are easier to develop and to expand.

If a closer look is taken a distinction can be made between XPS and Mars. A clear distinction is that XPS is in a further development than Mars. XPS has many resources available for developers, while Mars has little or almost no resources for development. The community of Mars is inactive, while the community of XPS is growing. Another important aspect is that XPS is officially released as an official format, whilst Mars is uncertain whether it will be released in the near future. Adobe has not make any intentions to make announcement whether Mars will go into a next phase of development. This uncertainty makes Mars less ideal for the Albumprinter format. XPS also has another advantage over Mars. The printing industry and the graphic market are more and more accepting XPS as a printing format. Hewlett Packard, the printer supplier of Albumprinter, already have printers that support XPS. Mars has due its uncertainty for now no support of the printing industry. As conclusion, XPS is for now the best candidate as the new Albumprinter format and therefore the best format to go to the prototype phase.

3 DESIGN

The design chapter covers two different designs:

- XPS File Format for Albumprinter
- Software prototype

The first paragraph will discuss the XPS file that will be used for the prototype. It will show how the XPS works and how this attributes to its use within Albumprinter. The paragraphs afterwards will be about the prototype software to create the XPS file in cohesion with the X-editor. First the design goal is made clear. Then certain design issues are described. The object model that is used for creating the XPS is presented thereafter. A full specification of the prototype can be found in the appendix.

Issue:

- XPS only describes the image only for printing without editing the image itself. Therefore it is possible to keep the original picture at all times, with for example transparency, cropping and resizing done within the XML description of the image.
- Integration X-editor means that the XPS module must be able to run from the X-editor environment. Due to the refactoring taking place while the prototype module is being created, the decision is made to use the legacy XML used by the X-editor before refactoring as the information provider of the XPS module. This means that an own model of the photo book needs to be created for use for the XPS module only. This is only for the proof of concept of using XPS as a photo book representation, while in the end there should be a common object model for the X-editor itself and the integrated XPS module.

3.1 MICROSOFT XPS

XPS stands for XML Paper Specification. This is a format developed by Microsoft as an answer in the market to Adobe's PDF as the new generation paper specification/file format. Its target is to become the digital paper of the future. Therefore Microsoft has been pushing this format in the printing industry, getting support from established printers like Hewlett-Packard and Fuji-Xerox.

XPS is a XML based paper specification as stated in the name. The choice to make a new paper specification XML based is not a surprise. XML has developed into a general markup language as it is human readable and therefore easy to edit and manipulate, while maintaining structure in the information it contains. The XPS-file is a compressed zip-file containing XML descriptions and resources like images and fonts. It consists out of several XML documents; together it describes the structure of the XPS. The XML files are spread throughout the file and it is structured in the way that the document specification file specifies what pages are attached, whilst the page specification file specifies the images and texts locations. These files are laid out in a structure recommended in the XPS document specification, but are not obligatory (Microsoft, XPS Specification and Reference Guide, 2006).

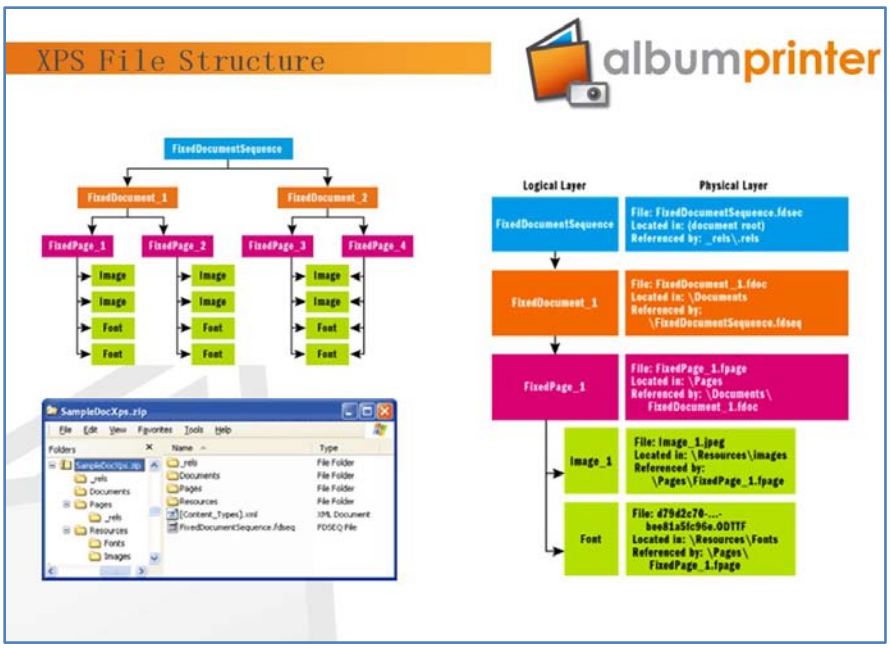


Figure 2: XPS file structure

Although most specifications are made within the XML, documents in the XPS-file there are a few files compulsory in a XPS document. For example there is a XML document describing all the file-types that are used within the XPS-file.

Also photos and fonts that are used in the document must be embedded in the file. This is due to the fact that XPS is by design a paper format, like PDF from Adobe, so all the information that are needed for printing must be contained within the file. Only this way the guarantee can be given that the printed document is equal to the screen presented version, this is what is known as the “What You See Is What You Get (WYSIWYG)” feature. Inherently for the same reason it is also an invalid XPS file when there are no fonts attached while text exists, as font information is crucial for any text representation.

Furthermore XPS is designed to separate content with presentation as much as possible. The images within the XPS are therefore unchanged while you can still add properties to the representation of these images within the XPS-document. XPS offers a wide range of properties for image processing on top of the image by description in XML in order to maintain the image unchanged. Features such as cropping, clipping and creating opacity are made possible in the XPS-document. By keeping the image to its original, storage space can be saved in contrary to the alternative method of modifying the image for each time that it is needed and only put the changed image in its place in the lay out, and therefore storing each image separately (Microsoft, XPS Specification and Reference Guide, 2006).

Summary:

- XML based
- What You See Is What You Get
- Fonts are embedded
- Images are embedded
- Image processing on by description

3.2 XPS-AP

XPS-AP is the XPS document created for Albumprinter for use with the prototype. It will be further referred to as the XPS file. This XPS is a specification adapted to show better semantics for use with a photo album. The actual XPS can be changed back into the standard XPS recommendation or other adaptations can be made to better reflect products for Albumprinter solutions. As it is shown in figure <XPS-design.jpg> there aren't any structural changes made to the file specification, actually two names of directories have changed to reflect that changes are possible, and to create a better understanding of the structure and its representation. *Documents* has been changed into *Products*, *Images* are now called *Photos.* The resources are stored for each *Product* so same resources can be shared throughout different pages in a logical fashion. In such a structure more albums or products can be stored in one XPS file. This can be a handy fact when the printer's RIP can handle XPS files directly.

Images in the XPS are stored in the *Photos* directory. As all the dimensions in XPS are given in points we have to take not that XPS uses a 96 points per inch representation. The reason for pointing this out explicitly is that PDF and most other formats use 72 points per inch representation. If this is not considered, the end result could differ from what is intended. In the prototype not all functions of XPS will be demonstrated, only the basic functions for text and image processing needed to represent a simple album will be demonstrated using the prototype. More complex capabilities, such as mirroring, masking, clipping and other view modifications of the images are not demonstrated in the prototype. More information is available in the XPS specification paper found in the appendix.

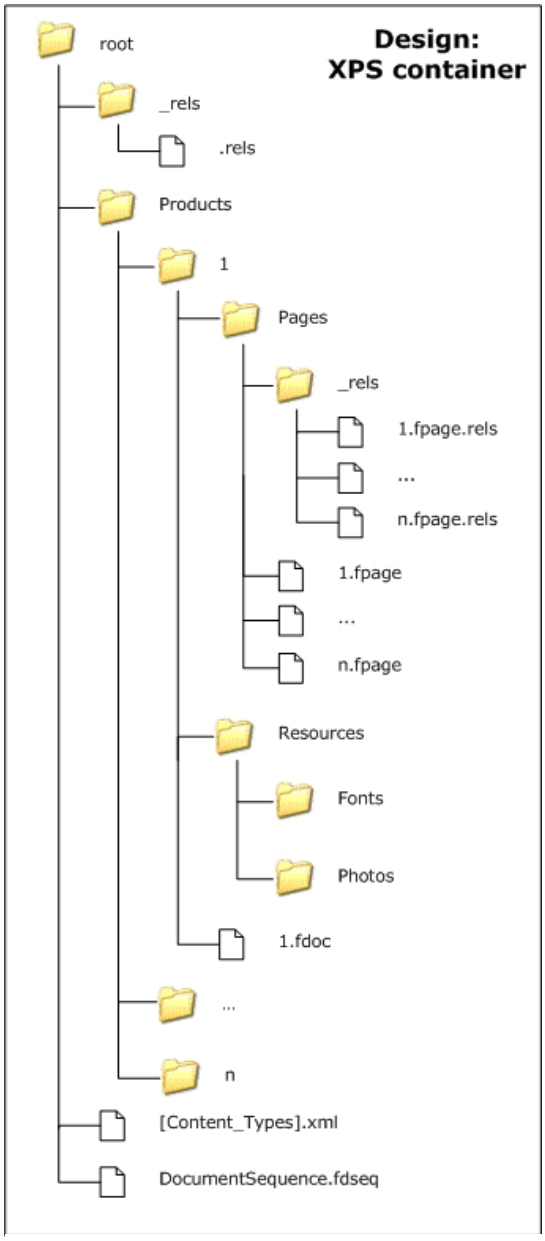


Figure 3: XPS tree

3.3 SOFTWARE PROTOTYPE

3.3.1 DESIGN GOAL

In short the design goal of the prototype is to show that XPS is capable of being the file format to carry photo book information. The prototype needs to show that within the XPS specification we are able to build software around it in order to create a legit XPS file representing a physical photo album. It must be able to provide the proof of concept or the fail in order to provide information needed on deciding the continuity of trying XPS as the new file format.

The system to be constructed will be a separate module to be added to the Albumprinter X-editor. The Albumprinter X-editor is being refactored as the XPS prototype is being made, hence the choice is made to use the old XML generated by the X-editor before refactoring as the source of information of the book. This XML document is an X-editor file that Albumprinter uses internally to contain the information of the photo album made with the X-editor and is a part of the software. Therefore the use of this XML for our prototype to form a XPS representation of a photo album is legitimized. As the XPS-prototype is being made, this is the best option to have the most complete information about the photo album.

3.3.2 OBJECT MODEL

First of all objects are created to represent the photo album. The information is extracted from the XML file located on the disc. This XML is generated by the X-editor when the user interface of the XPS prototype application is loaded. All of the information will be made in to attributes and objects. This leads to the object model for an album. However this is different than the object model the X-editor after refactoring will be using, this model is developed independently of the refactoring. The case here is that purely for the XPS the object model doesn't have to contain as much information as the X-editor might need for the application, and as such for the XPS-prototype application it is sufficient to use the simplified model.

3.3.3 GRAPHICAL USER INTERFACE

As a prototype the user needs to be able to control the loading of the XML source and check/adapt the values of the model. The necessity is shown in the fact that this application is a prototype to test for the fact whether a XPS can be successfully constructed to represent a book. In other words there are two things that are experimental here, the construction of the object model out of the source-file and secondly the construction of a valid XPS representing the object model. In order to provide flexible testing a GUI will be constructed to show all the values loaded from the XML and made editable before creating a XPS document, while leaving the XML source-file unchanged.

3.3.4 CONTROLLER

A separate controller class is constructed to handle the events coming in from the GUI. This controller is then able to construct the object model from the XML source. This controller will then also be able to construct the XPS file from this object model. Also any modification to the object model is done by the controller. In general the controller is fed a XML source-file and then a XPS is generated.

3.3.5 DIAGRAMS

The UML diagrams and the sequence diagram of the design above can be found in Appendix C5.

4 IMPLEMENTATION

In this chapter the process of constructing the XPS-AP is described in the first section. It consists out of 3 steps which are elaborated. Afterwards the implementation of the prototype application for constructing this XPS-AP is given, including the issues that are uncovered. Where possible solutions for these issues will be provided.

4.1 XPS-AP

As the main focus of this project is on the function of XPS, this chapter starts with the explaining of the XPS-AP implementation. The creation of an XPS-AP has the following steps: First the main structure of the XPS-AP is build [4.1.1], following by the creation of the pages within the XPS-AP file [4.1.2] and embedded in the pages the implementation of the images [4.1.3] and texts [4.1.4]. And the final step is zipping the XPS-AP as an XPS-file [4.1.5].

4.1.1 BUILDING XPS-AP STRUCTURE

First, the main structure of the XPS-AP container is build. As designed in the section 3.3, all required folders and files are created. These files and folders form the main structure within the XPS-file. These required files contains references to files within the XPS-AP for building an valid XPS.
The following files are created during this step:

- ***required*** Document reference file: *.rels*

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Target="/DocumentSequence.fdseq" Id="R0"
                Type="http://schemas.microsoft.com/xps/2005/06/fixedrepresentation"/>
</Relationships>
```

Figure 4: example .rels

Figure 1 shows an example of the document reference file. This file has a reference to the document sequence file. This is done by the <*Relationship*> property.

The reference to the document sequence has to be valid, otherwise this will lead to a corrupt XPS-file.

- ***required*** Fixed Document Sequence file: *.fdseq*

```
<?xml version="1.0" encoding="utf-8"?>
<FixedDocumentSequence xmlns="http://schemas.microsoft.com/xps/2005/06">
    <DocumentReference Source="/Products/1/1.fdoc"/>
</FixedDocumentSequence>
```

Figure 5: example .fdseq

Figure 2 shows an example of the fixed document sequence file. References are made to the fixed documents in the XPS file by the *<DocumentReference>* tag.

This reference to the fixed document has to be valid, otherwise this will lead to a corrupt XPS-file.

- ***required*** Fixed Document file: *.fdoc*

```
<?xml version="1.0" encoding="utf-8"?>
<FixedDocument xmlns="http://schemas.microsoft.com/xps/2005/06">
    <PageContent Source="/Products/1/Pages/1.fpage"/>
    <PageContent Source="/Products/1/Pages/2.fpage"/>
    <PageContent Source="/Products/1/Pages/3.fpage"/>
</FixedDocument>
```

Figure 6: example .fdoc

Figure 3 shows an example of the fixed document file. References are made to the fixed page files in the XPS file by the *<PageContent>* tag. Depending on how many pages, one or more references have to be declared here.

If a reference to a page is missing, the page will not be shown in the final result.

- ***required*** Fixed Document file: *[Content_Types].xml*

```
<?xml version="1.0" encoding="utf-8"?>
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">

    <!-- REQUIRED -->
    <Default Extension="rels" ContentType="application/vnd.openxmlformats-package.relationships+xml" />
    <Default Extension="fdseq" ContentType="application/vnd.ms-package.xps-
                                                                fixeddocumentsequence+xml" />
    <Default Extension="fdoc" ContentType="application/vnd.ms-package.xps-fixeddocument+xml" />
    <Default Extension="fpage" ContentType="application/vnd.ms-package.xps-fixedpage+xml" />

    <!-- FONTS -->
    <Default Extension="odttf" ContentType="application/vnd.ms-package.obfuscated-opentype" />
    <Default Extension="ttf" ContentType="application/vnd.ms-opentype" />

    <!-- IMAGES -->
    <Default Extension="png" ContentType="image/png" />
    <Default Extension="jpg" ContentType="image/jpeg" />

</Types>
```

Figure 7: example [Content_Types].xml

Figure 2 shows an example of a content type file. The filename has to be *[Content_Type].xml* and it should located in the root-folder. This file contains all file-type declarations, which are used in the XPS file. Different file-types as schemas, fonts and images can be declared in this file. Declarations are done by the *<Default>* tag.

Missing declaration leads to a corrupt XPS-file. It's important to declare all file-types that are used in the XPS-file.

During the implementation of this part, it was clear that the declarations needs to be implemented according to the XPS specification. An incorrect reference will lead to an invalid XPS-file. In Appendix C some samples can be found of these XPS errors.

4.1.2 CREATION OF PAGES

After the main structure of the XPS-file has been build, pages are created and added to the XPS container. In XPS this is described by a marked-up language in the fixed page files; *.fpage*

For every newly created page, references has to be updated in the fixed document file, as described in the previous section [4.1.1].

- ***required*** fixed page file: *.fpage*

```
<FixedPage xmlns="http://schemas.microsoft.com/xps/2005/06" xml:lang="en-us" Width="816.37"
Height="1058.26">

    <-- empty page -->

</FixedPage>
```

Figure 8: example .fdoc

Figure 3 shows an example of an empty page. A page is described as a fixed page with the *<FixedPage>* tag and is saved as an *.fpage* file in the XPS container. Required attributes for this tag is the Width and Height of a page, and the defined attribute of the XPS-schema.

Nested in the *<FixedPage>* tag images, text and other graphical representation of a page can be described. These description used the properties of the defined schema of XPS. For a full list of properties see the XPS Reference Guide (Microsoft, XPS Specification and Reference Guide, 2006).

- ***required*** fixed page reference file: *.fpage.rels*

Every fixed page file has a reference file. This file is required and contains references to the resources, as images and fonts, that are used for that specific fixed page file. The following figure shows an example of this page reference file.

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Target="/Products/1/Resources/Photos/9244.JPG" Id="P0"
                                    Type="http://schemas.microsoft.com/xps/2005/06/required-resource"/>
    <Relationship Target="/Products/1/Resources/Fonts/Arial.ttf" Id="F0"
                                    Type="http://schemas.microsoft.com/xps/2005/06/required-resource"/>
</Relationships>
```

Figure 9: example .fpage.rels file with references to resources within the XPS file

4.1.3 ADDING IMAGES

Images are described in the fixed page file within the XPS container. For the prototype the following features are thoroughly explored:

- Placement of images
- Rotation of images
- Opacity of images
- Image operations: scaling, rotation

These features of XPS is shown in the demonstration files of the XPS package, and can be downloaded from the Microsoft website (Microsoft, XPS sample documents, 2007).

Image description are embedded in the page files. Figure 2 shows an example of an image description in the fixed page file.

```
<Path Data="M -11.3376,-11.3376 L 827.7216,-11.3376 L 827.7216,568.7136 L -11.3376,568.7136 Z">
        <Path.Fill>
                <ImageBrush ImageSource="/Products/1/Resources/Photos/9244.JPG"
                        Viewbox="0,0,734.4,491.52"
                        ViewboxUnits="Absolute"
                        Viewport="-26.8512,-11.3376,870.0768,580.0512"
                        ViewportUnits="Absolute"
                        TileMode="None"
                        Opacity="1"/>
        </Path.Fill>
</Path>
```

Figure 10: Image representation example

In XPS the images are placed in a certain area called a *Path*. Paths can then be filled with a *ImageBrush* element in which we identify the image source location in absolute or relative to the page file within the XPS. Then a *Viewbox* is created, this viewbox is that part of the image that you want to show in the file. This could be a small part of the image or the photo in its whole. The dimensions are given in XPS points. Now the Viewport is defined. The viewport will show the image we mapped from the viewbox. This viewport can be set in anyway, but remember that the path will overrule the viewport when the viewport is bigger than the path dimensions. Opacity is valued at 1 to show no transparency. The value of opacity is ranged between 0 and 1.

In the prototype is shown that images can be implemented successfully while meeting certain conditions. For example when images contain the so-called EXIF information it can get a bit tricky, when pointing to the Viewbox this has to be considered due to the fact that the ImageBrush works with height and width in point defined in the image by using the dimension in pixels information and the resolution in dpi information within the EXIF information. If not however, XPS will take use the default 96 pixels per inch when reading the image. For the prototype we have chosen to use only images containing EXIF information and the dpi valued at 300.

Also not every element of *ImageBrush* or any other *Brushes* that exist in the XPS are used in the prototype. For instance, we have decided not to use the z-index of the images for queuing the images from top to bottom. At the

moment we just take the images from the XML from in a FIFO manner, the first image that is listed is first placed in the XPS.

More of information on these options can be found in the specification. For showing the photo book defined in the XML in a XPS, it was sufficient to use the basic XPS elements.

4.1.4 ADDING TEXT

Text are described with properties and attributes defined by the XML-based XPS schema. The description of a text is embedded in the fixed page file within the XPS. The following figure shows an example of a text description in a fixed page file.

```
<FixedPage xmlns="http://schemas.microsoft.com/xps/2005/06" xml:lang="en-us" Width="816.374399" Height="1058.2656">

        <Canvas RenderTransform="1,0,0,1,274.8576,724.4352">
                <Glyphs OriginX="155.66719999999998"
                        OriginY="18.666666666666664"
                        FontRenderingEmSize="15.118079999999999"
                        FontUri="/Products/1/Resources/Fonts/Arial.ttf"
                        StyleSimulations="None"
                        Fill="#000000"
                        UnicodeString="example text">
                </Glyphs>
        </Canvas>

<FixedPage>
```

Figure 11: example text representation

A decision is made to use a Canvas, wherein the text will be placed. The canvas functions as a text field, which will support matrix transformations, like rotation and other rendering transformations. The description of the text is described within a *<Glyphs>*-tag. In this tag several attributes can be defined for the representation of a text. The following attributes in the *<Glyphs>*-tag are the basic elements to describe a simple text:

- ***OriginX*** and ***OriginY***: the starting coordinates of the text in the canvas.
- ***FontRenderingEmSize***: the size of the text
- ***FontUri***: reference to the font that is used for the text
- ***StyleSimulations***: styles applied to the text. For example bold and italic.
- ***Fill***: the color of the text
- ***UnicodeString***: a Unicode string of the text.

This is only a small subset of attributes for representing a text in XPS, for a complete list of properties and attributes, see the Reference Guide of XPS.

By design of XPS, fonts should be placed in the XPS container, the attribute *FontUri* should point correctly to the font-file within the XPS. It's also important to update the reference file of the page, which was discussed in 4.1.2. If one of these references is incorrect, the XPS will be corrupt.

During the implementation the following problems arise for representing texts:

1. Basic styles like bold, italic are available in XPS, while underlining is not a style attribute in XPS. Although in XPS it's possible to represent underlining. XPS also have graphical operations, like drawing lines, circles and other graphical operations. So, it's possible to implement an algorithm with the graphic operations to visualize underlining for text.
2. Aligning text is not a basic operation in XPS. As XPS is a fixed page description, text should be defined precisely according the coordinate system of XPS. Therefore aligning cannot be seen as a basic operation in XPS.

If possible, solutions for the problems above will be given in the next chapter.

4.1.5 ZIPPING

In order to finalize the creation of the XPS-AP, all files and folders have to be zipped in one single zip-container. This zip-container has as extension *.xps*. The whole structure of the XPS with all its references and XML files are compressed and save as an XPS file with the open-source library FZip (Wahlers & Herdenker, 2008). FZip is an ActionScript 3 class library for editing, modifying and loading zip-archives. The created XPS file, can be viewed in an XPS-viewer. And because it's a zip-archive, it can also be opened in every regular zip software.

4.2 PROTOTYPE APPLICATION

In this second section of this chapter, the prototype application will be discussed. First the environment of the application [4.2.1] is described, following by the implementation of the object model [4.2.2] and the XPS control [4.2.3]. Afterwards the Graphical User Interface [4.2.4] is discussed and this section ends with the integration of the prototype application with the X-editor [4.2.5].

4.2.1 ENVIRONMENT

The environment in which the prototype application will be working is the Adobe FLEX environment [ref]. The prototype will be used as a component by the X-editor application.

Adobe FLEX is an open framework for building interactive web applications. The core language of Adobe FLEX is ActionScript 3. By also using MXML, an XML-based markup language, FLEX offers an easy way to build and lay out graphical user interfaces.

As none of us were familiar with FLEX it took some time to get use of this environment. After all it was a good and interesting environment to work in.

4.2.2 OBJECT MODEL

First the object model is implemented in this phase. The object model is a representation of an album, as designed in the previous chapter.

During the implementation of the object model the issue arise that the XML provided by the X-editor is complex and not all data from the XML is needed for creating an XPS. It was a challenge to extract all the relevant data for

the object model during the implementation phase. The XML was full of undefined data, which was addressed to the X-editor development team to take in consideration for their future work.

At the end the object model is implemented with all attributes of the XML. A decision was made not to make a distinction during this phase, due it was unclear which data was exactly needed for the prototype.

4.2.3 XPS CONTROL

The next step in the implementation phase is to create a controller which will use the object model to build and edit a XPS file. In this section only the main methods with relevance of the working of the XPS file will be discussed.

4.2.3.1 LOADXML(XML)

This method creates an object model from a XML file.

As in a early stage the XML module is not integrated with the X-editor yet, this method reads a local XML ,which is essential for testing purposes. Till the prototype is integrated with the X-editor no real-time XML can be provided by the X-editor therefore a local XML has to be used till that phase. The local XML file is a standard pre-created XML file from the X-editor containing a valid description of an album with several photos and texts.

During the implementation of this method it was uncovered that the XML provided by the X-editor contains massive data and the structure was very unclear. With support of the X-editor team relevant data was extracted from the XML and whereby needed solutions was provided to improve the XML of the X-editor.

4.2.3.2 TOXPS()

This function creates an XPS file from the object model.

When an object model is available an XPS can be created. This function is the main focus of this project. And therefore the functionality of this method is separately discussed elsewhere in this chapter. The construction of this method was discussed in detail in section 4.1. Problems during the implementation of this method can also be found in that section.

4.2.3.3 LOADXPS()

Loading a local XPS file into the object model.

This was an extra function to show that XPS can be used in the other direction as well. It was to find out whether an XPS provided by a 3rd party could be loaded into an object model, which represent an album. After a quick research it is plausible to read a 3rd party XPS, many conversion has to be made. And as references several other software applications already provide these conversions. Due to time issues, and on agreement with the supervisors, this functions is not implemented.

4.2.4 GRAPHICAL USER INTERFACE

The GUI of the prototype is designed with Adobe FLEX. This GUI is a visual representation of the object model and has controls to call the functions of the XPS control. The following figure shows the interface.

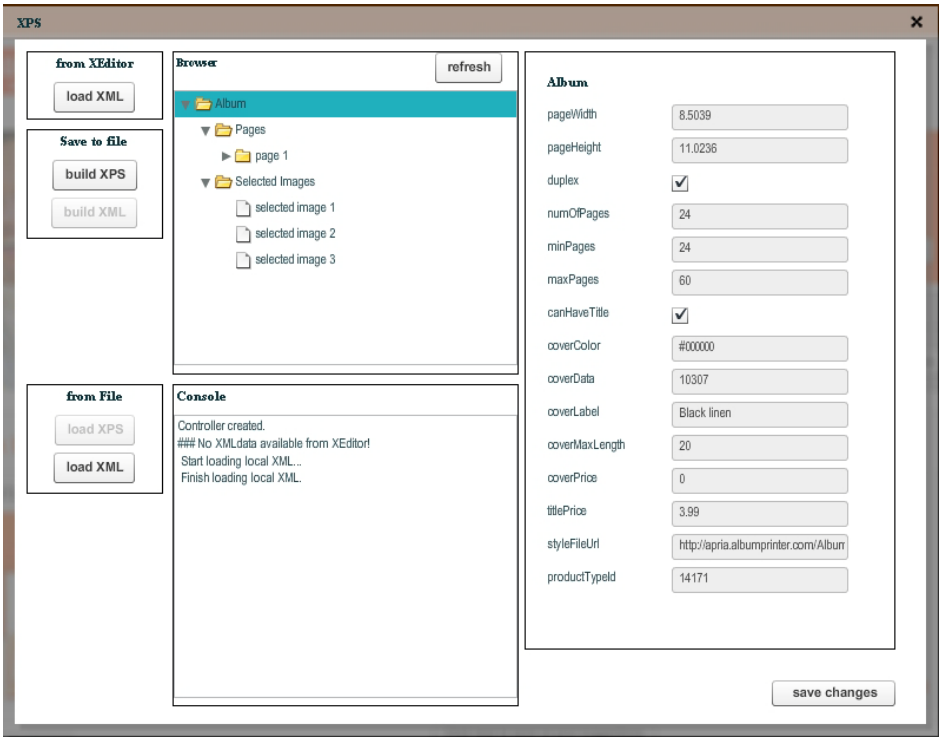


Figure 12: Prototype interface

It was interesting to use the GUI and layout functionality of the FLEX framework. FLEX provides a wide range of GUI/Layout components for building up GUIs from sketch. In combination with the ActionScript programming language FLEX provides a complete tool for designing interactive GUIs.

4.2.5 INTEGRATION WITH X-EDITOR

As a final step in the implementation phase, the XPS software prototype is integrated with the X-Editor. The X-editor application use the prototype application as a component. As the application is only for prototype purposes the prototype screen is only shown in debug mode.

After completing the integration with the X-editor, an important functionality is then made available in the XPS software prototype. The function to load an XML directly from the X-editor is then available. With this functionality real-time data of a photo album can be derived from the X-editor for creating a representative album in XPS.

5 CONCLUSIONS

After the implementation there are several outcomes. The results are presented in this chapter and where problems arise solutions will be given if possible.

5.1 XPS STRUCTURE

The XPS structure as designed in 3.3 is a good representation of an album at Albumprinter. Although this design is working properly, the following points need to take in consideration at the time XPS will be use in the Albumprinter applications:

- XPS specification has a fixed system of attaching pages in a package. A choice had to be made whether to only insert one or more albums per XPS file.
- As following from section 4.1; A decision has to be made which non-relative album information is necessary to store in the XPS.

Above all, the XPS structure is a good representation of an album for X-editor. Although, there are some final decisions need to be made, the basis of XPS structure as designed in chapter 3 is good.

5.2 IMAGE

Certain issues arise when images are added to the XPS in the prototype. Most of the issues occurs due to the fact that XPS uses a 96 points per inch setting while the standard is 72. Images themselves can contain or not contain "resolution" information in dpi.

Conversions need to be made before an image can be loaded and mapped to the correct area.

```
// big container view port
var imgL: Number = image.getCpX()                          * alb.XPS_POINTS_PER_INCH; //Container whole photo x coord
var imgW: Number = image.getCpWidth()                      * alb.XPS_POINTS_PER_INCH; //Container whole photo right coord
var imgT: Number = image.getCpY()                          * alb.XPS_POINTS_PER_INCH; //Container whole photo y coord
var imgH: Number = image.getCpHeight()                     * alb.XPS_POINTS_PER_INCH; //Container whole photo bottom coord
// piece that is shown on the page (data field)
var imgPosL: Number = image.getX() * alb.XPS_POINTS_PER_INCH;// Position left (in xps points)
var imgPosT: Number = image.getY() * alb.XPS_POINTS_PER_INCH;// Position top
var imgPosR: Number = imgPosL + image.getWidth() * alb.XPS_POINTS_PER_INCH;// Position right
var imgPosB: Number = imgPosT + image.getHeight() * alb.XPS_POINTS_PER_INCH; //Position bottom

//TODO: Check for Exif and TIFF information in case of jpg or tiff. If it exists
//use it to determine the dpi in the file information, if there is none, use the standard 72dpi (96/72=4/3)
var dpi: int = 300; // standard used "dpi" when there is no indication in the image property
```

Figure 13: conversion dpi code

In the end the prototype demonstrates that it is very plausible to use XPS for a photo album from the image point of view. Because of the XML it is relatively easy to understand what happens to the image in the file and to apply the WYSIWYG. There is a possibility to apply all kinds of effects within the XPS specification without changing the original image, which indeed lead to storage efficiency and file safety (Image can always be restored to original).

5.3 TEXT

Representation text in XPS is relatively simple. XPS provides a XML-based language, which describe text in a readable and understandable way. This language has a wide-range of functionality to represent text for a fixed document file.

There are several problems, which has to be solved first:

1. Underlining of text
   This problem can be solved with the use of the graphical operations in XPS. XPS provide graphical operations to draw lines, circles and other graphical functionality. With these operations it's possible to create an algorithm to realize underlining for texts. Further research has to be taken to tackle this problem. It's clearly not impossible, as other software application already implement this functionality with XPS.
2. Alignment of text
   As XPS is a fixed page specification, it doesn't support the basic functionality of alignment of text. An algorithm has to be created for calculation the precise coordinates for the text input. This problem is not unfamiliar at Albumprinter. In previous applications it already had tackled this problem. And since this problem for XPS doesn't differ from the current problem with the current file format, it's definitely possible to solve this problem.

Overall, text representation in XPS have no major issues, which are not solvable. Therefore the conclusion is that XPS provides all functionality to visualize text for the Albumprinter file format.

5.4 CONCLUSION

Looking at the prototype results and our problem statement,

"Is there an open file format that can replace the current proprietary format without loss of quality of the product and extensibility for expansion?"

Following from the results in this chapter the answer is: Yes, there is an alternative file format that can replace the current file format used by Albumprinter. However this has only been preliminary and further tests are needed to ensure the effective use of XPS as replacement for the current proprietary format.

6 RECOMMENDATIONS

There are still several things to consider when handling images in XPS. The way the Viewbox works on a photo using the EXIF information may lead to the need of standardizing the image dpi output to a certain value to make sure that no surprises occur in the XPS file. This means that the X-editor should either catch the possible error or that it provides the XPS generator with the correct dpi value to use during generation.

Background-images are also something that doesn't exist in the XPS prototype yet. However this could be easily resolved by thinking of the image as the lowest layer of image on a page. As XPS won't handle it like objects, it doesn't matter for XPS whether it's an background-image or just another photo.

Covers are special pages that need to be implemented in the XPS generator too. These will be pages that have odd sizes relative to the normal page size.

Texts on the other hand still have some more complicated issue to consider. It mainly has to do with the fact that, again, XPS is a paper specification tool and sees text as just a special picture. Things like lines do not exist in XPS, therefore line spacing, text alignment and things like underlining are special text features that need to be implemented and calculated before passing it to the XPS. E.g. XPS see text underlined as a piece of text with a separate line drawn under it.

PDF conversion has been done during prototyping using Adobe Acrobat Professional 8.1 Trial Version. It shows that it is possible to make the right conversion. Commercial tools, like NIXPS (NIXPS, 2008), exists also for command-line conversion to PDF, if it is possible to create a PDF using the Adobe Acrobat command-line then this is where the preference is.

A. BIBLIOGRAPHY

[1] Adobe. (2006). Retrieved july 15, 2008, from Adobe Mars Project: http://labs.adobe.com/technologies/mars/

[2] Ecma. (2006, december). Retrieved july 15, 2008, from Ecma Office Open XML overview: http://office.microsoft.com/en-us/products/HA102058151033.aspx

[3] Microsoft. (2003). Retrieved july 15, 2008, from Microsoft XPS: http://www.microsoft.com/whdc/xps/default.mspx

[4] Microsoft. (2007, july). Retrieved july 2008, 15, from XPS sample documents: http://www.microsoft.com/whdc/XPS/XpsSamples.mspx

[5] Microsoft. (2006). *XPS Specification and Reference Guide.* Microsoft.

[6] NIXPS. (2008). Retrieved july 15, 2008, from NIXPS: http://www.nixps.com/

[7] OASIS. (2003). Retrieved july 15, 2008, from OASIS Open Document Format for Office Applications: www.oasis-open.org/committees/office/

[8] PODi. (2008). Retrieved july 15, 2008, from PPML: http://ppml.podi.org/

[9] Wahlers, C., & Herdenker, M. (2008, february). Retrieved july 2008, 15, from FZip: http://codeazur.com.br/lab/fzip/

B. TABLE OF FIGURES

C. APPENDIX

C1. RESEARCH THESIS

OPEN FORMAT SHIFTING TO XML

July 1 2008,
An Ngo, Xinan Zhu

ABSTRACT

In this thesis open file formats in the Graphics Arts market will be discussed in relation with the bachelor project at Albumprinter. Recently more and more open file format standards are available. Open file format is a published specification for storing digital data, which is maintained by a non-proprietary standards organization, and they are free of use. The trend is that more and more companies will use open standard file formats for storage of the data of their applications. And these formats tends to make use of XML description to provide a more open structured usage of documents.

INTRODUCTION

The current situation with file formats for the Graphics Arts market is that these formats are not open in the way that stored data as images and text are not retrievable for reuse or editing. Currently new formats are designed with the usage of XML to solve this problem. The main research question in this thesis is "Why are open file formats in the Graphics Arts market shifting to the usage of XML?".

First this thesis start with a background section about the current situation at Albumprinter and why they are looking for a new file format. Then it continues with a section about the current situation of the file formats in the overall Graphics Arts market. Follows by a section about the upcoming of new file formats with XML and then it will end with a conclusion, where the research question will be answered.

BACKGROUND

Albumprinter B.V. is a software company that provides services for creating personalized photo-products, mainly books, calendars and organizers. For these customized products Albumprinter stores relevant digital data, such as images and text.

In order to do this Albumprinter B.V. provides the next branded software consumer packages: a Windows editor and a cross-platform X-editor. As shown in figure 1, Albumprinter B.V. provides a whole life-cycle; from client software to production. It has a section where software are developed and a production facility where the photo-products are printed and shipped to the customers. It also provides client services such as helpdesk and services for resellers.

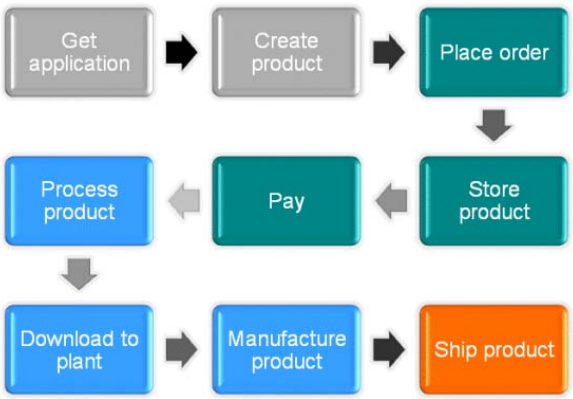


Figure 1 Albumprinter process

Since the digital photography has been introduced there is a big market for photo books software as the ones

provided by Albumprinter B.V.. Due to the digital camera more photos are taken, but they are digital instead of the old well-known photos on paper and pasted in photo books. Still, people are more attracted to the physical products like real paper photo books. Albumprinter B.V. provides this service: shoot photos with your digital camera, create and make your own photo book and it produces your personalized photo book. Last year a total of 1 million photo products have been produced and shipped to the customers.

THE CURRENT FILE FORMAT IS HARD TO SUSTAIN FOR ALBUMPRINTER

At Albumprinter currently digital photo-books and albums are stored in an own proprietary format during editing. This is converted to PDF in order to print it. However once converted to PDF it is hard to do editing on the file. This has lead to the development of the own .ALB file format in the first place. However having an own proprietary format leads to several problems.

First of all by having an own format means that all the development and tools needs to be created and maintained by Albumprinter. There are no 3rd party tools available and there will not be. Secondly the format itself needs to be updated by Albumprinter too in order to be adequate in the support of their activities. Thirdly as Albumprinter tries to open up the platform for resellers it is essential that it is easy accessible and the learning curve should be short for developers to use the Albumprinter API. Right now with the .ALB format the problem of not having a flood of documentation arises for the developers and 3rd parties, and this leads to extra unnecessary costs and time could be saved by using a standard format.

The goal for Albumprinter is to find a new file format for their applications to replace the current .ALB format. This is necessary to be able to provide an open and easy API for the reseller clients of the company, furthermore it will save costs made on maintenance of the own proprietary format and shorten the learning curve for developers using the file format. It is also the goal of the company to use this new format in their complete set of applications.

GRAPHICS ARTS MARKET

The graphics arts market, is the market which provides services of printing productions, from software products to full service printing. This industry uses complex file formats. The file format in the graphics arts industry stores different kind of data, such as data of images, text and graphics. Albumprinter belongs to this industry. Their format needs to store image-, text- and graphics data.

File formats that are used in this industry are Page Description Languages (PDL). PDL describes the data of documents in a higher level than the lower level of the data-bit at printing [1]. This high level data are readable by printers and RIP stations. PDF and PostScript are PDL's, which are globally used in the printing industry [2].

An advantage of PDF and PostScript is that they describe precisely the data on printed pages. In other words *"What You See Is What You Get"*; what you see on the screen, will be exactly print on paper. This is a very important aspect in the printing industry; there should be no difference between the data on the screen and on the paper after printing.

A disadvantage of PDF and PostScript is that they are non-open file formats. The data are stored in such a way that data of objects, such as images and text are hardly to retrieved. Reusing and editing data in these formats are therefore not possible.

NEW DPL'S WITH XML

PDL's as PDF and PostScript have the disadvantage of non-retrievable data. New PDL's are now available or in developing without this disadvantage. PDF is shifting towards the use of extensible markup language (XML), which will provide more structured data storage, without loss of specific graphical data [3].

In 2003 Microsoft announced to develop the file format XPS that will compete with PDF [4]. XPS is currently available and has the same benefits as PDF and in advantage, data are retrievable for reuse. In 2006 Adobe launches the project Mars, as a file format successor of PDF [5]. XPS and Mars are both comparable and they both have a promising future for the printing industry. Both file formats use XML for its data description and has the same benefits as PDF [6], [7].

In table 1 Mars and XPS are compared with PDF. The aspects used in the following table is derived from the requirement for the Albumprinter file format. The WYSIWYG aspect is clearly importance for a file format in the graphic industry, which was discussed earlier in this thesis. The second aspect that the format should be open will provide usage of 3rd party collaboration and it would provide less maintenance of the own proprietary format. The third aspect follows from the second aspect. XML-based is a rising language for open file formats. The sustainability of the format is also important, as the format should retain for the coming years. Reusing of content/layout is an aspect that would provide easier development of the software. And the last section is whether the printing industry is supporting the file format.

Format Specifications	Mars	XPS	PDF
Document fidelity (WYSIWYG)	✓	✓	✓
Open specification	✓	✓	x
XML based	✓	✓	x
Sustainability	x	✓	✓
Reuse content / layout	✓	✓	x
Print / RIP support	**Mars**	**XPS**	**PDF**
HP	x	✓	✓
Fuji Xerox	x	✓	✓
Océ	x	?	✓

Table 1: Comparison Mars, XPS and PDF

The main advantages of Mars and XPS over PDF is the reuse of the data content. Both new PDL's are open formats in contrast with PDF and they both use the open format XML. The potential of XML is to improve links with data and to allow open source software to connect with open documents [8]. This last aspect is the big advantage over PDF; the reuse of the data is hereby supported. Mars and XPS looks promising to be a good candidate for Albumprinter's new file format.

CONCLUSION

More and more Open File Formats in the Graphics Arts industry are shifting to the use of XML. The advantage of the usage of XML, lies in the potential of XML to improve links with data and allow open software to use these data. The new formats, as Mars and XPS have the same benefits as currently widely used PDF and PostScript in the printing industry. XPS and Mars are good alternatives as the new format for Albumprinter. Both new formats have a great potential to replace PDF and PostScript in the near future. This new trend of shifting file formats to use XML is in an early developing stage. The Mars format is still in a very early phase of developing, while XPS is already used by some organizations and (printing) companies, but it is not that far developed and not totally embraced by the printing industry yet to replace the current formats PDF and PostScript.

References

[1] Nik Stanbridge, *Page Description Language Document Coversion and Manipulationi*, 2007

[2] Paul Collin, *Analysis: XPS and the Graphics Arts Market*, February 2007

[3] Thor Olavsrud, *PDF format Shifting to XML*, http://www.internetnews.com/dev-news/article.php/2235261, July 13 2003

[4] Official XPS website, http://www.microsoft.com/whdc/xps/default.mspx, June 2008

[5] Official Mars website, http://labs.adobe.com/technologies/mars/, May 2008

[6] Mark Vruno, *What's the status: Microsoft XPS vs. Adobe PDF?,* http://www.graphicartsonline.com/blog/1840000384/post/1480013348.html, July 13 2007

[7] Dany Amiouny, *White Paper: PDF or XPS: Choose the Right Document Format for your Applications*, 2007

[8] William Pollard, *Mars Opens PDF to XML Connections*, http://english.ohmynews.com/articleview/article_view.asp?article_class=4&no=372521&rel_no=1, July 19 2007

C2. PLAN OF ACTION

Plan of action

New albumprinter file format

Bsc-project TU Delft

An Ngo
Xinan Zhu

Amsterdam, 7 April 2008

Table of Contents

1 INTRODUCTION

Before we start the project at Albumprinter we made this Plan of Action. This document will be a guideline thorough our project.

2 ASSIGNMENT DESCRIPTION

2.1 INTRODUCTION

In this chapter we will describe the assignment. First we will describe the Employer, the contact persons. Then we will describe the problem statement and the assignment description and the deliverables of this project.

2.2 THE EMPLOYER

Albumprinter B.V. is a software company that provides services for creating personalized photo-products, mainly books, calendars and organizers.

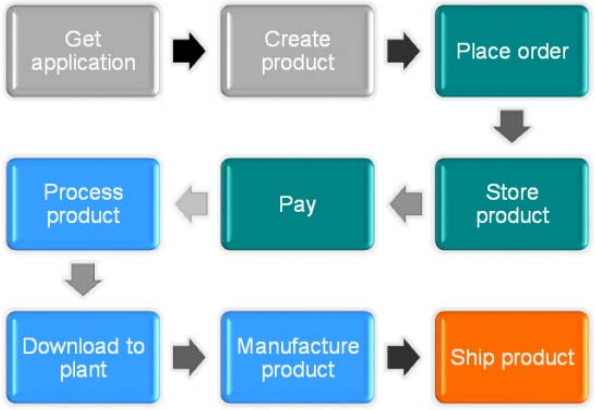


Figure 1 Albumprinter process

In order to do this Albumprinter B.V. provides the next branded software consumer packages: a Windows editor and a cross-platform X-editor.

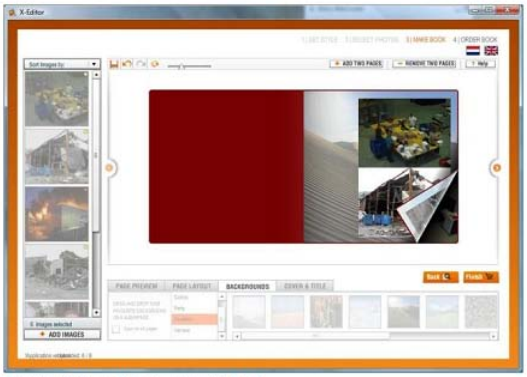


Figure 2 X-editor screenshot

Figure 3 Windows editor screenshot

As shown below Albumprinter B.V. Provides a whole life-cycle; from client software to production. It has a section where software are developed and a production facility where the photo-products are printed and shipped to the customers. It also provides client services such as helpdesk and services for resellers.

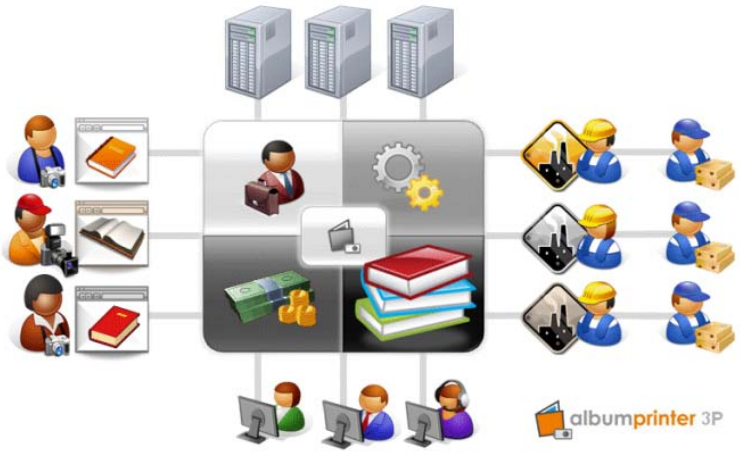


Figure 4 Albumprinter high level architecture

Since the digital photography has been introduced there is a big market for photo books software as the ones provided by Albumprinter B.V.. Due the digital camera more photos are taken, but they are digital instead the old well-known photos on paper and pasted in photo books. Still, people are more attracted to the physical products like real paper photo books. Albumprinter B.V. has this service: shoots photos with your digital camera, create and make your own photo book and it produces your personalized photo book. Last year a total of 1 million photo products have been produced and shipped to the customers.

2.3 CONTACT PERSONS

TU Delft
Dr.ir. C.A.P.G. van der Mast
LS Mens-Machine Interactie
C.A.P.G.vanderMast@tudelft.nl
Elektrotechn., Wisk. & Inform.
Room : HB 10.120
Mekelweg 4
2628 CD Delft
Office: +31 (0)15 2782549

Albumprinter B.V.
Dhr. K. Waslander
CTO Albumprinter B.V.
klaas@albumprinter.com

Stationsplein 55
P.O. Box 14606
1001 LC Amsterdam
Office: +31 (0)20 5218950

2.4 PROBLEM STATEMENT

The current .ALB file format is hard to sustain for Albumprinter.

At Albumprinter currently digital photo-books and albums are stored in a own proprietary format during editing. This is converted to PDF in order to print it. However once converted to PDF it is hard to do editing on the file. This has lead to the development of the own .ALB file format in the first place. However having a own proprietary format leads to several problems.

First of all by having a own format means that all the development and tools needs to be created by Albumprinter. There are no 3rd party tools available and there will not be. Secondly the format itself needs to be updated by Albumprinter too in order to be adequate in the support of their activities. Thirdly as Albumprinter tries to open up the platform for resellers it is essential that it is easy accessible and the learning curve should be short for developers to use the Albumprinter API. Right now with the .ALB format the problem of not having a flood of documentation arises for the developers and 3rd parties, and this leads to extra unnecessary costs and time could be saved by using a standard format.

2.5 GOAL

Finding a new file format for the Albumprinter applications to replace the current .ALB format. This is necessary to be able to provide a open and easy API for the reseller clients of the company, furthermore it will save costs made on maintenance of the own proprietary format and shorten the learning curve for developers using the file format. It is also the goal of the company to use this new format in their complete set of applications.

2.6 ASSIGNMENT

The assignment here is to build a prototype application that applies the newly chosen file format, which shows that it can do exactly the same as the current file format. The new file format to be tested is chosen within the first 2 weeks of the assignment. Afterwards development of the application will commence. This application will show that it is possible to open a file containing a page with multiple objects (photos), perform a simple edit (mirroring some images and moving it to another location on the page and saving the file in the same file format, printing it to the printer.

2.7 DELIVERABLES

The project will encompass:

1. Set of requirements for the format
2. Standard selection report
3. Prototype application
4. Documentation for application

3 APPROACH

3.1 INTRODUCTION

In this section we will describe the approach of this project. It also includes a schedule of the project and a Gannt chart.

3.2 WORK

First we have to do a research on format files that are suitable to replace the current format file at Albumprinter. Then we have to implement the format file, create an prototype application and integrate our prototype with the current editors. And at the end the project we have to write a report and give a presentation of what we our findings.

3.3 SCHEDULE

The schedule of the project from 7 April 2008 to 1 July 2008

Phase	**Date**		**Duration**
1. Research	7 April	18 April	2 weeks
Delivery of Plan of action	11 April		
2. Implementation	21 April	30 May	7 weeks
2.1 File format + testing	21 April	2 May	2 weeks
2.2 Application + testing	5 May	16 May	2 weeks
2.3 Integration + testing	19 May	30 May	2 weeks
2.4 Finishing + testing	2 June	6 June	1 week
3. Report	9 June	1 July	3 weeks
Delivery of Report	1 July		
Presentation	T.B.D.		

For this schedule we also made a Gannt chart. This can be found in the Appendix A.

The report phase is compared to the other phases very long. In this period we have exams that's why we have planned more time for this phase than usual.

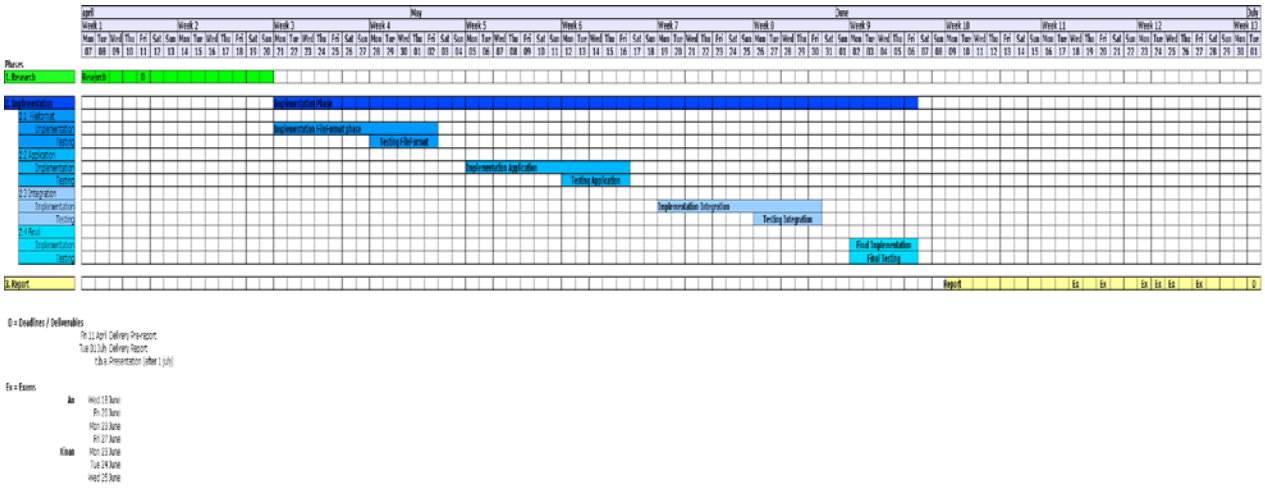
Exams

An:
18/06
20/06
23/06
27/06

Xinan:
23/06
24/06
25/06

APPENDIX

APPENDIX A: GANNT CHART



C3. TEST RESULTS

In the research phase, some basic features of XPS were explored. The following figure shows a test result of a basic XPS, containing 5 small pages with basic features.

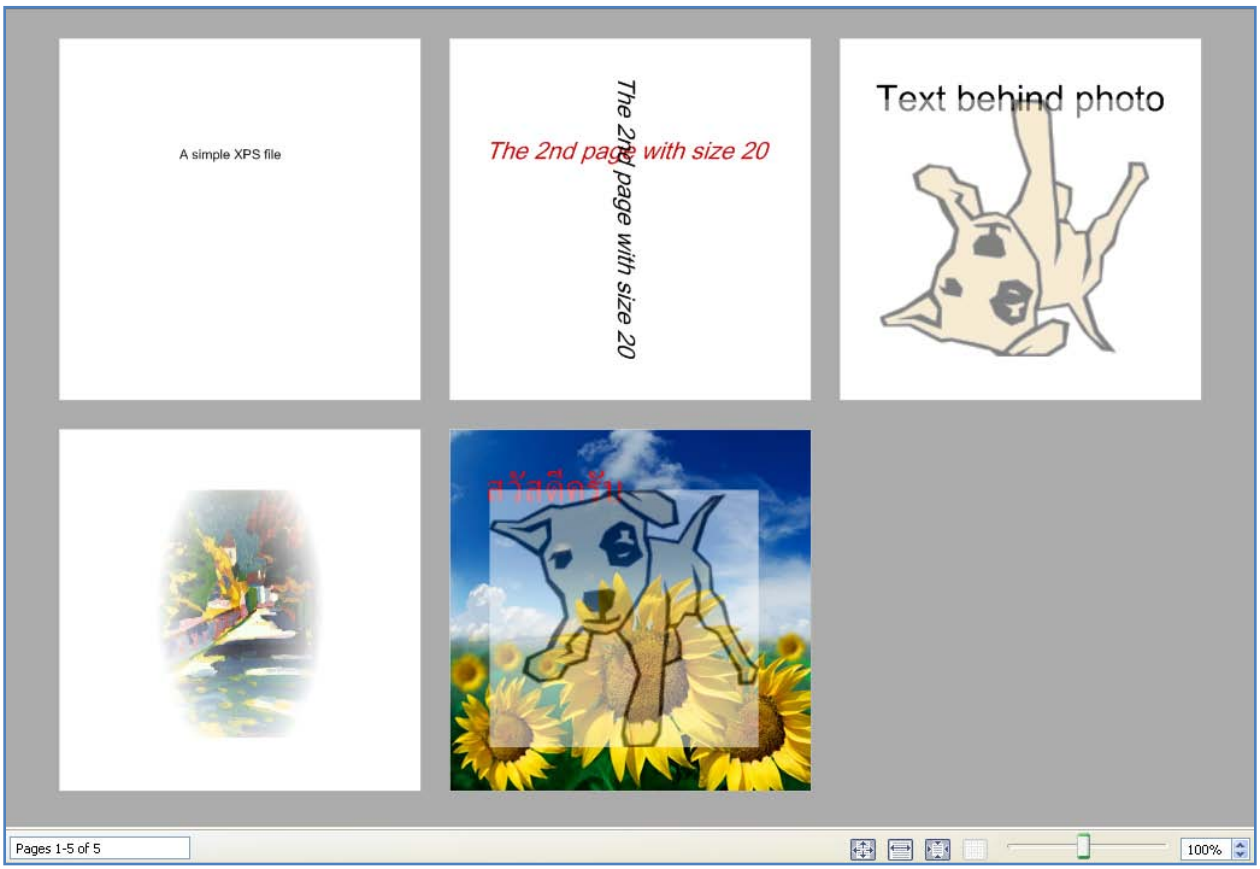


Figure 14: example XPS features

Page 1: a simple text
Page 2: rotation of text
Page 3: text behind image, image is transparent
Page 4: oval mask over an image
Page 5: unicode text, background image and transparent image in front.

C4. COMPARISON TABLE

A comparison of potential file formats for Albumprinter.
● = supported
□ = poorly supported
? = unknown

We did a research on 4 XML-based extensible format and 1 printer language / job description format (PPML):
- Adobe Mars
- Microsoft XPS
- Sun/OASIS ODF
- Microsoft OOXML
- PODi PPML

General File Format	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
XML based	●	●	●	●	●
Sustainability	□	●	●	●	●
Portable / Multiplatform	●	●		●	●
Support by printing industry	?	●	●		
License agreement (Royalty-free)	?	●	●	●	●
PDF conversion	●	●	●	●	●
Graphics Capabilities	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
Support for transparency photos	●	●	●*	**	**
Support for CMYK color space	●	●	●*	**	**
Support for alpha channel (opacity)		●	●*	**	**
Printing	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
Ensure document fidelity (WYSIWYG)	●	●	●		
CMYK support for professional printing	●	●	●		
Retains print job information deferred or remote printing		●			
Support for crop boxes and bleed boxes needed by professional printers	●	●			
Hewlett Packard RIP support		●	●		
Fuji Xerox RIP support		●	●		
Océ RIP support		?	●		

Appendix C4: Comparison Table

Security	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
Password protection against unauthorized access	●	●	*	●	
Password protection against document modification	□		*		
Digital signature of full document	●	●	*		
Digital signature of specific parts of a document	●	●	*		
Image Support	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
JPEG (RGB and CMYK)	●	●	●	●	●
JPEG 2000	●				
JBIG2 for grayscale images	●				
PNG images	●	●		●	●
CCITT for black and white images	●	□			
TIFF images (RGB and CMYK)		●	●	●	●
HD photo		●			
SVG	●		●****		
Color Space Support	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
RGB	●	●	●	●	●
CMYK	●	●	●		
Gray	●	●	●		
N-Channel	●	●			
ICC Profiles	●	●	●		
Named and Spot colors (e.g. Pantone)	●	□			
Font support	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
TrueType fonts	●	●	*	***	***
Type1 (Postscript) fonts	●		*		
CFF (Compact Font Format or Type2)	●	●	*		
Vector fonts	●		*		
Support for multi-national character sets	□ Unicode only	●	*	□ Unicode only	
Full font embedding	●	●	*		
Partial font embedding	●	●	*		
Protecting an embedded font from being extracted from the document	●	□	*		

Appendix C4: Comparison Table

Document Structure	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
Document bookmarks and outline	●	●	*	●	●
Support for page thumbnails	●	●	*	●	
Support for editable form fields	●	□	*	●	
Support for annotations	●	□	*	●	
Editing Document Content	**Mars**	**XPS**	**PPML**	**ODF**	**OOXML**
Original resources can be retrieved	●	●	*	●	●

*** n/a as PPML is a language not a file format (and does not embed files in a zip-container)**
**** n/a as ODF/OOXML are not page description / layout formats**
***** Unspecified, depending on system (CSS2)**
****** PPML allows SVG resources. It is possible to use SVG.**

C5. DIAGRAMS

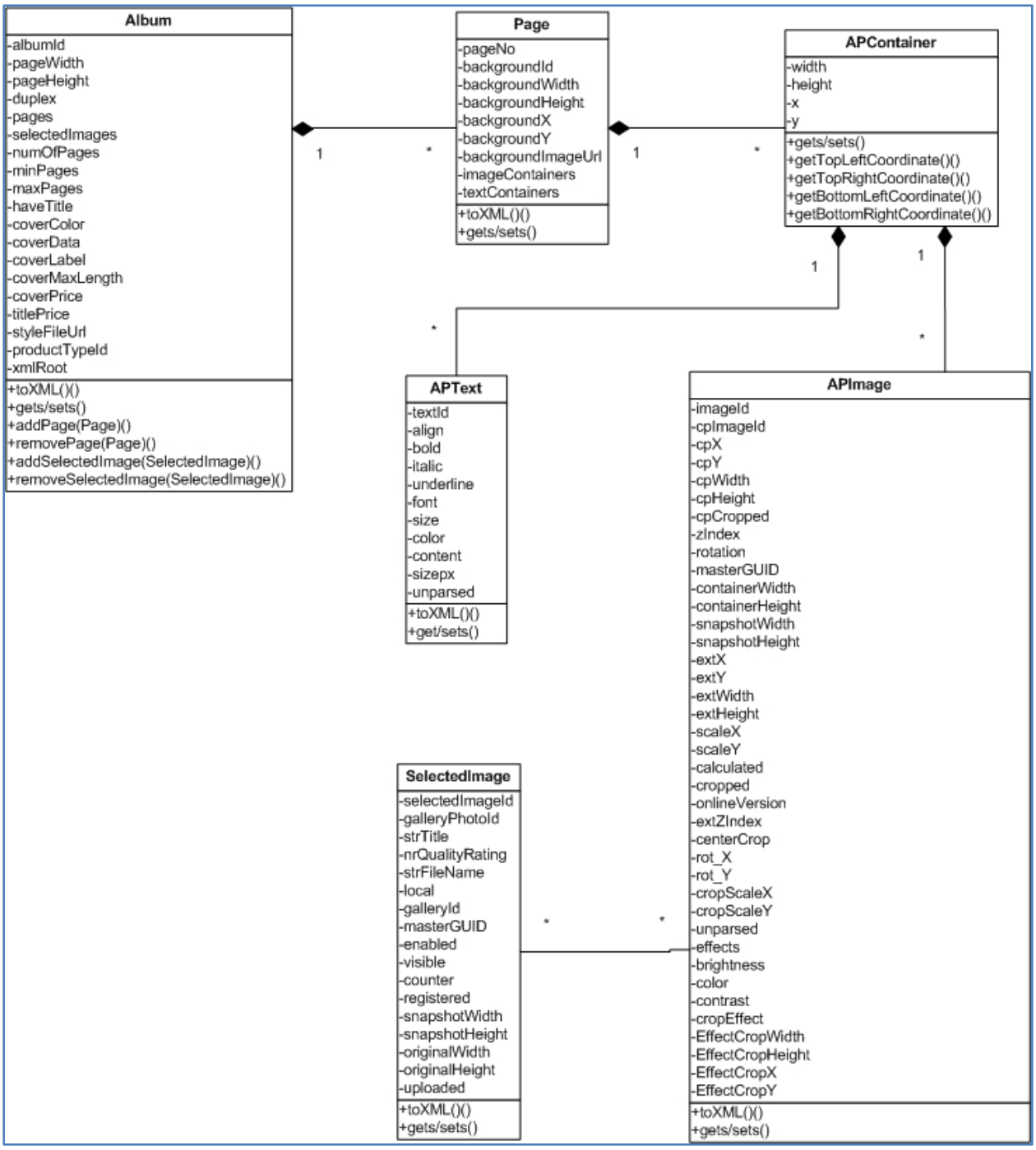


Figure 15: UML Diagram of the album Object Model

Appendix C5: diagrams

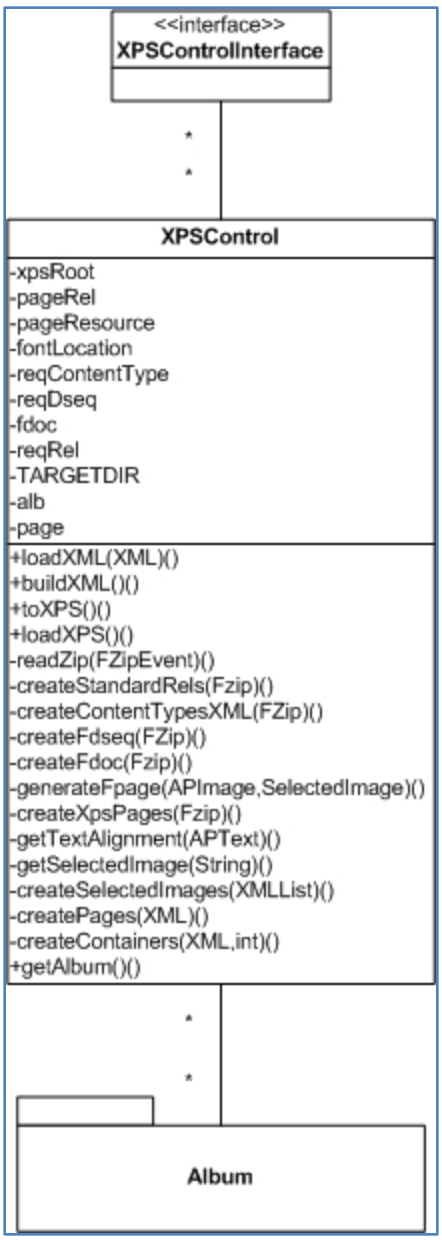


Figure 16: XPS controller diagram

Appendix C5: diagrams

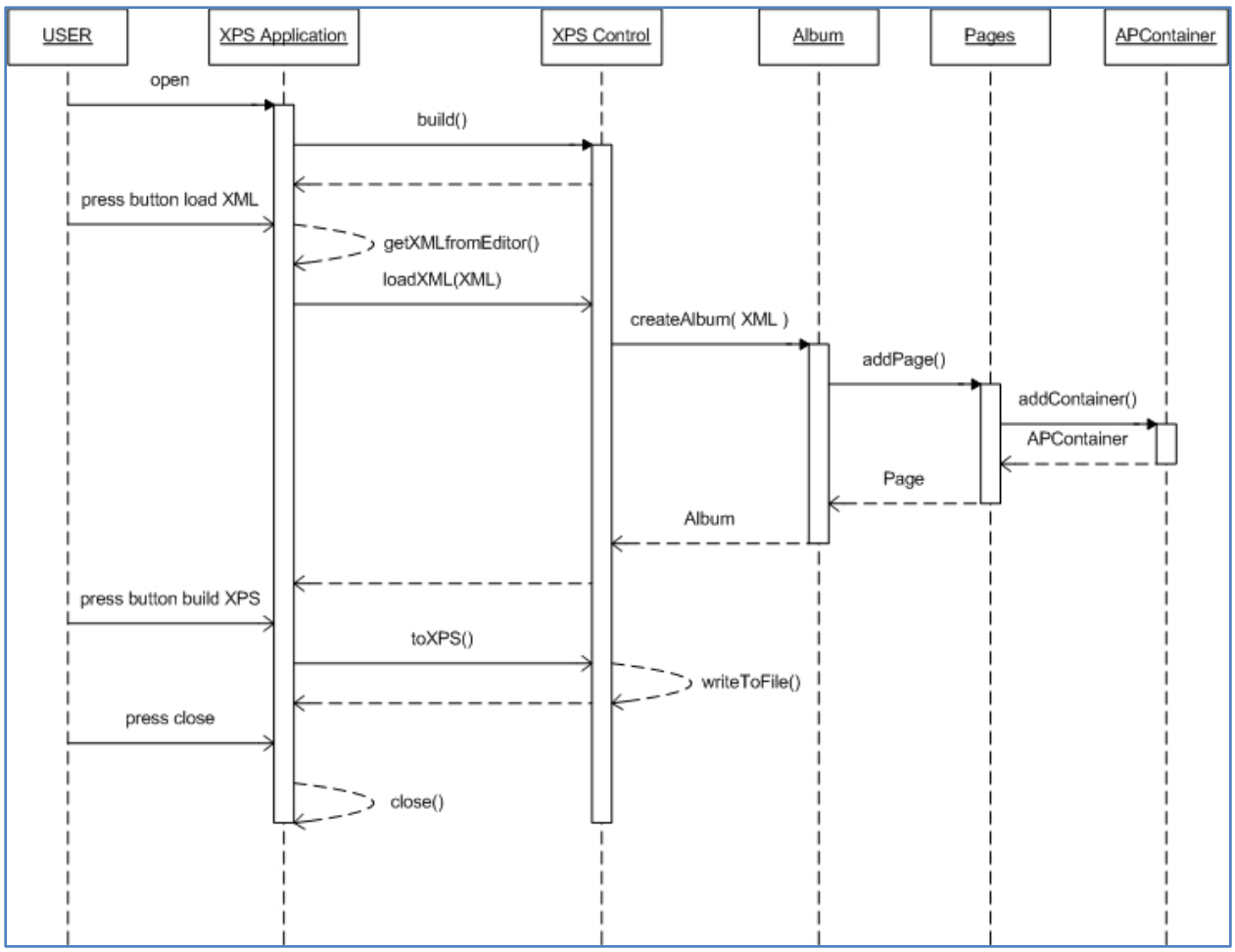


Figure 17: Sequence diagram