Postscript Finishing Feature Controller

43

Prepared By
Muhammad Azeem

Supervised By
Abdul Qudus Abbasi

Institute of Information Technology

Quaid-i-Azam University Islamabad

Session 2010-2011

STATEMENT OF SUBMISSION

This is to certify that **Mr. Muhammad Azeem** Registration. No. 01161011003 has successfully completed the final project as "**Postscript Finishing Feature Controller**" Quaid-i-Azam University, Islamabad to fulfill the partial requirement of the degree "**Master of Science in Information Technology**".

10-07-2012

External Examiner

Dr. Naveed Akram
Associate Professor

Faculty of Computing,
Riphah International University,
I-14, Islamabad.

Internal Supervisor

Mr. Abdul Qadus Abbasi
Assistant Professor

Institute of Information Technology
Quaid-i-Azam University, Islamabad.

A Report Submitted to the
Institute of Information Technology,
Quaid-I-Azam University Islamabad,
As a partial Fulfillment of the Requirements
for the Award of the Degree of
Master in Information Technology

Project in Brief

- ***Project Title***
  Postscript Finishing Feature Controller
- ***Under Taken By***
  Muhammad Azeem
- ***Supervised By***
  Mr. Abdul Qudus Abbasi
- ***Organization***
  Elixir Technologies pvt. Ltd.
- ***Started***
  September 2011
- ***Completed***
  January 2012
- ***Software Tool***
  Visual Studio 6.0
  Notepade++
- ***Operating System***
  Windows Xp
- ***System Used***
  Intel Centrino 1.8MHz
  Ram 1GB
  Hard Disk40 GB

Dedicated to

My father (late), mother, sister and my teachers

for their endless support, affection, trust and encouragement

Abstract

This is a Postscript Printer finishing feature controlling application. In software engineering, this is an example of software customization.

This application takes two files as input i.e. Postscript printer description (PPD) and Postscript (PS). Application parses PPD file extract file header and printing option information from file, store in data repository and displays options to the user in tree view form. Application gets input PPD file and checks data repository. If file has already parsed then its information will not store in database and system will simply loads information to GUI and let the user further processing. User selects printing finishing options according to his choice. Selected option may or may not have command against it. Then postscript file is parsed and PS parser gives the location where command is to be updated. Updater gets these locations reference and commands. Finally, postscript file will be updated accordingly. Only those printing options can be updated in PS file that are supported by PS file.

Acknowledgement

ALLAH has been the greatest source of strength for me and I am humbly thankful for the blessings He has conferred upon me. My **parents** and **sister's** love, trust and support have always played a major role in the achievements I have blessed throughout my life.

I am obliged to pay my sincere and heartiest gratefulness to my supervisor or more accurately my counselor, **Mr. Abdul Qudus Abbasi**, for he was the biggest motivation behind initiation of my project. His timely guidance and useful suggestions not only helped in this software building but also in my character building.

I would like to present my sincere thanks to **Mr. Abu-Bakar** (my external supervisor) for his true guidance and motivation to fulfill requirements of this project.

I would like to pay my sincere thanks to all my teachers (Ms. Madiha Haider Syed, Ms. Sidra Batool Kazmi, Ms. Abida Sadaf, Ms. Robina Rashid, Mr. Khurram Gulzar Rana and Dr. Azhar Saeed), for they taught me very informative and interesting courses that proved worthy to improve my skills. I am also thankful to the staff of Institute of Information Technology, Quaid-i-Azam University for their friendly attitude.

I am very grateful to **Mr. Abdul Khaliq** (cousin) and other relative for their sincere prays, greedless support and encouragement.

To all my friends, and my class fellows, thank you for your understanding and encouragement in my many moments of crisis. Your support motivated me for all the time.

This thesis is a First step of my journey.

Muhammad Azeem

Table of Contents

List of Figures

Chapter No. 1

Introduction

1.1. Introduction

Computer has spread in all walks of life; it might be field of education, industry, government or private sectors, or entertainment. Information technology is implemented in all fields in some shape. We cannot deny the worthiness of Information Technology. Our lives has surrounded by illusion of Information Technology.

Printing documents has become very essential part of our daily life necessities we use computers to write our documents, assignments, research papers, books, journals, agreement papers and reports etc. After that we use printer and print our documents on paper. So, it is an easy way to write above mentioned items in computer instead of manual writing with pen. It saves the time and we can make changes in document easily and can remove errors with less cost. When we select the printing option available in menu bar we get option pane window for selecting printing. These options are page type, number of pages, range, printer model, color or black etc. These options are available by default.

Human beings have been keen in getting more and more control on things either Allah's made or human self-made. This project is also an attempt to get more control and access on printing finishing features. I want to add printing features in PS file in my own way according to my desire. A large number of printers follow postscript and also called postscript printers. **Postscript** is page description language that tells the information about the page to printer hardware. All information of printing options is kept in a file that is called **postscript printer description** (PPD) file.

Windows based computer systems communicate with the printer by means of a printer driver and PPD files. Computer that has windows operating system can also communicate with the printers by means of a Printer Control Language (PCL), printer driver and PDD files (the PCL equivalent of PPD files). Both enable to use special features of the printer from the print dialog box.

All communication between application and printer is managed by printer driver. Printer driver interprets the instruction generated by the application, Printer specific options (set by user) and

instruction generated by application are merged, and then translates all information into Adobe Postscript or PCL (print control language). It is a language that all postscript printers understand. In simple word, printer driver write a postscript file based on original file (PPD) and the options we set from the user interface (i.e. Print dialog box).

1.2. Purpose

The basic purpose of this project is to provide more control on existing postscript file (Elixir Specific) with respect to printer controller. It will convert an Elixir postscript file to a more efficient printer controller specified file based on postscript printer description (PPD) file. We have postscript printer description (PPD) files and postscript (PS) files. Software will read postscript printer description (PPD) file and update postscript file according to PPD file. This application will be used in print related industry.

1.3. Scope

Postscript printer description files are important sources of printing features. But it is very complex and difficult to read due to its specific format. PPD files have their own commands and to understand it is very time consuming. There are multiple companies that are manufacturing postscript printers and have their own PPD. It is very difficult for user to read complete file and extract printer options from them but user wants to print file and want to insert command manually or to choose options according to its requirements instead of selecting all options. For this purpose, to overcome this complexity there should be a mechanism that provides a solution to the users in terms of features extraction. We need automated support to extract printer feature information from PPD file and visualize it in such an efficient way that it is more understandable to users.

C++ is considered a core language that has both functions oriented and object oriented features. It is very powerful language that gives great flexibility. That is why development of my project postscript finishing feature controller is based upon C++ language. I shall parse the postscript printer description (PPD) file and extract mandatory information and discard all unnecessary data, generate tree view of finishing features for ease of users. The Tree View will have key i.e. name of the option and values against that option and also command for to execute that option. By this, users will get necessary information of features and can select or deselect according to their choice.

Summary

In this chapter, introduction of the system is given. Application will be used in print related industry. Further, purpose and scope of system is also elaborated in this chapter and also in next chapter.

Chapter No. 2

Requirement Analysis and Specifications

2.1. Introduction

Requirement analysis is first step of software development life cycle. Requirements are the things or functions or more accurately services that should be delivered by the software (system).

2.2. Requirements Types

There are two main types of software requirements:-

2.2.1. Functional requirements

These requirements are related to functionality of the system. What functions are to be performed by system are considered as functional requirements. These requirements explain the system behavior.

2.2.2. Non-functional requirements

A Non-Functional Requirement is usually some form of constraint or restriction that must be considered when designing the solution [1].

The requirements are not about functionality but these are related to those elements that enhance the system performance and under these elements, system can perform functionality in better way. Further detail is next in this document.

2.3. Requirements Illustration Techniques

We have several requirements illustration techniques that are used to get requirements and explain them. Some techniques are as under.

- Interview
- Surveys
- Questioners
- Meetings
- Joint application development(JAD)

All the techniques are of great worth according to their use and these techniques have proved very beneficial in gathering software requirements. Every technique is used according to need and available sources. Mostly, I have used two techniques (i.e. meetings and interviews) commonly to gather required information from the organization. I constantly have meeting with organization concerned person and also meet other person who also deal and have knowledge about Postscript printer description and Postscript language.

2.4. Functional Requirements

Different functional requirements are as follows:

- System Input
- Reading and Parsing Postscript printer description file (PPD).
- Database Repository
- Creating tree view of functionality
- Reading and Parsing Postscript (PS) files.
- Update PS File

2.4.1. System Input

Postscript finishing feature controller System will take PPD and PS files as input. System's PPD section will get only PPD file and PS section will read only PS file as input and discard any other format.

2.4.2. Reading and Parsing Postscript printer description (PPD) file

As name PPD indicates it is postscript printer description file so it has detail of printing options, data, commands and other necessary information. Postscript printer description file has its own structure. My software application will get PPD file as input and there will be PPD file handling mechanism implemented at backend. PPD file handling mechanism will read the PPD file and pass it to tokenizer through an interface.

Tokenizer will split file into tokens there will be a token's collection now. This token's collection will be given to the **Paring mechanism.** This will be a customized PPD parser. There is a lot of data that is unnecessary or meaningless for us. So, we have to separate useful and useless data according to our need. For this purpose parsing is required. Parser will apply

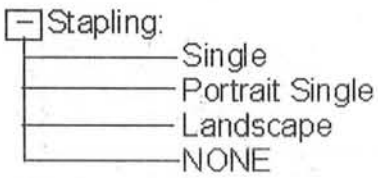
different filters on token collection and required data/information i.e PPD header and data information between OpenUI .... CloseUI, will be filtered. Collection will be used to save parsed data. Parsing will be done at backend.

2.4.3. Database Repository

There will be a database at backend where all parsed PPD file information will be stored. It is selectable. User can store parsed data into text file or in data base tables.

2.4.4. Creating Tree View of Functionality

When file parsing is done then user interface for this parsed data comes under discussion. How it will be seen to user? There are multiple option that is drop down list or tree view etc. Tree view is used to show this parsed information to the user. That means it will be visually represented by tree view structure. For example staple information is parsed and it will look like this.



This tree view will be selectable. When tree view is generated then option selection topic is under considerations. How user will select option according his choice? For this requirement I will provide checkbox to select and deselect option by checking and unchecking the option. It is very easy mechanism that every type of user can understand easily. User will have choice to select functionality according to its will. Application will provide default options as well as choice to update.

2.4.5. Reading and Parsing Postscript(PS) File

Postscript printer understand postscript file to print document. Postscript file is given to the system. Actually, this will be output file to be updated. Application will get postscript (PS) file as input and there will be PS file handling mechanism implemented at backend. This will read the PS file and pass it to tokenizer for further processing.

Tokenizer will split PS file into tokens and there will be a token collection now. This token collection will be given to the paring mechanism. This will be a customized PS parser. There is a lot of data that is unnecessary or meaningless for us. So, we have to separate useful and useless data according to our need. For this purpose parsing is required. Parser will apply different filters on token collection and required data/information i.e. header, page number, page level, document level etc. will be filtered. Collection will be used to save parsed data. Software will find the location where command is to update. Parsing will be done at backend.

2.4.6.Updating Postscript File

There is a Postscript command with options in PPD file. System will allow the user to select options and map the selected command into postscript file. There will a complete mechanism to update PS file according to selected options. That command combines with print control language (PCL) and then printer understand this command and perform task accordingly.

2.5. System Use Cases

A use case in software engineering is a description of steps or actions between a user and a software system which leads the user towards something useful [2]. The user might be a person or it may be a more abstract entity, such as external software.

Use Case ID	1
Use Case Name	Open PS Finisher Controller
Actor	User
Precondition	Application is ready to use
Post Condition	PS Finisher Controller is opened
Point of Extension	Finisher Controller
Normal Flow	1. User gives the command to system for opening PS Finisher Controller. 2. System displays the PS Finisher Controller.
Alternative Flow	1. system will show error message properly and closed

Use Case ID	2
Use Case Name	Input PPD File
Stakeholder and Interest	User wants to Input PPD file to application
Actor	User
Precondition	Application is ready to use
Post Condition	PPD File is opened/Loaded. System parses PPD file extract printing options and displays to user in tree view.
Point of Extension	PS Finisher Controller
Normal Flow	1. User selects browse button. 2. System displays a file dialog box to user. 3. User selects file and press open button or double clicks on file. 4. PPD file is input successfully.
Alternative Flow	1. The user selects file other than PPD file. i. System displays message "Invalid file". 2. User press cancel button. ii. System displays message "Invalid file".

Use Case ID	3
Use Case Name	Select Print Options
Stakeholder and Interest	User wants to select options from generated tree view by the PPD parser.
Actor	User
Precondition	System is in active condition. PPD file is parsed and printing options exist on screen in tree view.
Post Condition	Print option(s) is/are selected.
Normal Flow	1. User selects option from tree by clicking checkbox. 2. Options are selected.
Alternative Flow	1. Options cannot select. • System generate relevant error message.

Use Case ID	4
Use Case Name	Input PS File
Stakeholder and Interest	User wants to Input PS file to application
Actor	User
Precondition	Application is in active state. User is able to press browse button.
Post Condition	PS file input successfully
Point of Extension	PS Finisher Controller
Normal Flow	1. User select browse button. 2. System displays a file dialog box to user. 3. User selects the input file name and press open button or double clicks on file 4. PS file is input successfully.
Alternative Flow	1. The user selects file other than PS file. i. System displays message "Invalid file". 2. User press cancel button. ii. System displays message "Invalid file".

Use Case ID	5
Use Case Name	Select Apply On Options
Stakeholder and Interest	User wants to select apply on option
Actor	User
Precondition	Application is in active state
Post Condition	Apply on option is selected
Point of Extension	PS Finisher Controller
Normal Flow	1. Default option is already selected. 2. User selects radio button for apply on option from apply on group. 3. Option is selected successfully.

Alternative Flow	1. The option does not selected successfully. i. System displays message "system error".

Use Case ID	**6**
Use Case Name	Update Postscript File
Stakeholder and Interest	User wants to Update Postscript file according to selected options
Actor	User
Precondition	Application is in active state. PPD file is parsed and print options are extracted and viewed in a tree view. Print options are selected by user. PS file is selected as input and is parsed by the system.
Post Condition	Input postscript file is updated
Point of Extension	PS Finisher Controller
Normal Flow	1. User press UpdatePS button. 2. System mapped values. 3. Message displays that "PS file updated".
Alternative Flow	1. User doesn't select the PS File and press the UPdatePS button/command. i. System displays the message that "file is not inserted or options are not selected". 2. No print option is selected. i. System displays the message that "file is not inserted or options are not selected".

Use Case ID	**7**
Use Case Name	Close application
Stakeholder and Interest	User wants to close application
Actor	User
Precondition	Application is in active state
Post Condition	The application is closed.

Normal Flow	1. User press "EXIT" button. 2. System is closed.
Alternative Flow	1. No alternate option. System must close.

2.6. Non-Functional Requirements

As for as non-functional requirements is concerned it is also mandatory part of any software system. We cannot ignore the worth of non-functional requirements..

2.6.1. Efficiency

System should be efficient and self-explanatory. User interface must be very easy to understand application's functionality and operating method. It means easy to understand and easy to use for user.

2.7. System Hardware Requirements

The system on which this application will be developed must have at least

- 1.8 GHz processor
- 1 GB Ram
- 40 GB Hard Disk

2.8. Software Interface Requirements

Windows XP operating system will be used to fulfill software interface requirement.

2.9. Software Development Requirements

- Visual C++ 6.0 will be used as development tool.
- C++ is used as developing language with support of STL.
- Notepad++
- Adobe postscript viewer
- Object oriented approach will follow in the application development.
- Development Model is Scrum.

Summary

In this section, main functional requirements of the system are discussed and some non-functional requirements are also elaborated. Use cases for the system are explained. Software requirements and software development requirements are also described.

Chapter No. 3

Technical Background and Literature Review

3.1. Introduction

This project belongs to postscript file customization. Two main file are used i.e. postscript file and postscript printer description file. In this chapter, postscript file and its structure will be discussed. Similarly postscript printer description file and its structure will be elaborated.

3.2. What is Postscript?

Printer has its own hardware structure and drivers. When we print a file there may be a lot of text and graphics (pictures and images) on that file to be printed. **Postscript is a programming language** that is adjusted for printing graphics and text whether on paper, film, CRT or LCD. In short it describes page so it is also called page description language. The purpose of Postscript is to provide suitable and useful language that depicts images as well as text in a device.

Postscript is well known for its use as a page description language in print industry. The postscript language is designed to convey a description of virtually any desired page to a printer. It possesses a wide range of graphic operators that may be combined in any manner. It contains variables and allows the combining of operators into more complex procedures and functions. Postscript page descriptions are programs to be run by an interpreter. Postscript programs are usually generated by application programs running on other computers. However, many postscript printers, including the Apple LaserWriter, have an interactive state in which the user may program directly in postscript.

3.2.1. Before Postscript

Before postscript introduction in market, design of printer was made based on character printing. Multiple technologies were used for this job but most of them were using same attribute that is glyph. Physically, it seems that is was stamped like typewrite keys or optical plates and was difficult to change. Input text was given in ASCII. Later on dot-matrix printer were introduced in market, trend changed with increasing popularity of dot matrix printers. In dot matrix printers

characters were printed by means of series of dots. There exists a table inside printer that has font description.

Raster graphics printing was also introduced in dot matrix printers. Graphics were sent to the printer in series of escape sequences.

After that plotters were came into introduction. Vector graphics printing was left to special-purpose devices, called plotters. Almost all plotters did share a common command language; these were good for graphic printing but limited for anything other than graphics. Besides this, these were rare due to high cost and slow speed.

With the passage of time, degree of development in printing industry grows high and laser printer came into existence. It combines the best features of both printers and plotters. Like plotters, laser printers offer high quality line art, and like dot-matrix printers; they are able to generate pages of text and raster graphics. A much pretty feature of laser printers is, it makes possible to position high-quality graphics and text on the same page. We can print graphics, images and text on the same page. Postscript made it possible to fully exploit these characteristics, by offering a single control language that could be used on any brand of printer.

3.2.2. Origin of Postscript

Postscript is a product of Adobe Systems, Inc. Adobe Systems was formed in 1982 by Dr. John E. Warnock and Dr. Charles M. Geschke. The concepts of the Postscript language were seeded in 1976 when Dr. John Warnock was at Evans and Sutherland Computer Corporation. At that time John Gaffney, of Evans and Sutherland, was developing an interpreter for a large three-dimensional graphics database of New York harbour. Gaffney conceived the "Design System" language.

John Warnock then joined the Xerox Coporation's Palo Alto Research Centre (Xerox P.A.R.C) to work with Martin Newell. They reshaped the Design System into JaM (John and Martin) which was used for VLSI design and the investigation of type and graphics printing - culminating in Inter Press, Xerox's printing protocol.

In 1982, John Warnock left Xerox, together with Chuck Geschke, and founded Adobe Systems Inc. The name Adobe was taken from an Indian creek near to where John Warnock lived. Their aim was to build a dedicated publishing workstation and the final two-dimensional graphics handling product was named Postscript.

About the same time Steve Jobs, who had earlier founded Apple Computers, was looking for a solution for a high quality office printing system problem. Steve Jobs urged Adobe to develop a system to drive a laser printer. With the drop in price of memory, the first low cost laser printer engine from Canon, and a bit-mapped computer from Apple, the first postscript printer hit the market in 1985.

As hardware continued to improve, memory price continued to fall and with the appearance of powerful processors, such as the Motorola 68000, graphics applications continued to expand and became more widespread. Postscript was a mature product and was fully embraced by the industry, including the world's entire leading mainframe manufacturer's (such as Digital, IBM and HP/Apollo) and, possibly more importantly, the PC manufacturer's (such as Apple, IBM, Atari, Amiga and Acorn Archimedes).

3.2.3. Overview of Postscript Language

PostScript is also a computer language like other languages C++, JAVA etc. Originally it was developed by Adobe Systems Incorporated to process graphic information to digital laser printers. It is very powerful language that can be used for general programming as well as to express graphics images. It is famous for its flexibility and compaction. It can be learnt easily and we can produce high quality images, graphics and text by hand written program.

In this section overview of some important features will be discussed. This language is specially developed for complex graphics. It also handles letters as graphics sophisticatedly.

3.2.4. Model of Postscript Language

Postscript language model have some concepts that are sketched out as under:-

- Interpreted
- Stack based
- Dictionary
- Name
- Number
- String
- Array
- procedure

This language is interpreted and based on stack. Dictionary data structure is used that make it flexible and extensible. Postscript language is also a high level language that is interpreted, not compiled. One should have good knowledge to use it because interpreted languages require more attention and care than compiled languages. The beauty of postscript language is that it can increase efficiency and reliability of the software dramatically.

3.2.5. Data Structure and Dictionaries

A pretty feature provided by PS language is Dictionary. A is a structure in which data is stored in key-value mechanism. It also provides several standard data structures for example arrays, strings, files etc.

All of these data types are maintained as postscript objects. And dictionary is a single way to store PS object for reuse. "**def**" operator is used to make entries in a dictionary. This can be used for defining variable and functions and can also be used for complex data structures.

3.2.6. Stack

A stack is a data structure that has same input and output path. Objects are placed one on the top of another. Stacks are called last in first out data structures. There are many stacks that are used by PS interpreter e.g. operand stack, execution stack, dictionary stack and graphic stack. Operands are pushed on operand stack. And pop by interpreter. Postscript language is stack based and so we can say it is an exercise of stack manipulation. Many errors come due to incorrect use and less knowledge of stack manipulation. So it is must to have pretty knowledge in order to write program in postscript language.

3.2.7. Postscript Flexibility

Postscript is an extremely **flexible** language. Functions that do not exist, but which would be useful for an application, can be defined and then used like other postscript operators. Thus, postscript is not a fixed language within whose limits an application must be written, but is an environment that can be changed to match the task at hand. Pieces of one page description can be used to compose other, more complicated pages. Such pieces can be used in their original form or translated, rotated, and scaled to form a myriad of new composite pages.

3.2.8. Graphics Concept

To manipulate image, Postscript has few concept to that must be known before image processing in PS. That is as under:-

3.2.9.1. Device Space

This is the space that describes coordinates which are understood by the printer hardware. This measure of this space is done by means of pixels. That means it indicates resolution of page.

3.2.9.2. User Space

This is the space that describes the location of the points in coordinate. This system is same as co-ordinate system studied in school mathematics. These user space co-ordinates are converted into device space automatically by device space.

3.2.9.3. Current Transformation Matrix

Above mentioned transformation (i.e. user space coordinates to device space coordinates) is done through the current transformation matrix. This matrix is a 3 * 3 matrix by which user can rotate, scale and translate user space within device space. This is power of postscript.

3.2.9.4. Path

It is a collection of disjoint line segments and curves that exist on the page. It does not indicate actual ink. It is imaginary tracing over the page. User draws ink along the path by operators. (e.g. stroke, fill etc.).

3.2.9.5. Current Path

The actual path that postscript program create at the current time is called current path. It is created piece by piece.

3.2.9.6. Graphics State

Collection of various settings describes the graphic state. For instance the current path, the current font, and the current transformation matrix make up the graphics state. This state can be saved temporarily for later use. For this purpose gsave and grestore operators are used.

3.2.9. Postscript Programming

It is pretty easy to write program in PostScript. Operands are pushed on operand stack and invoked to use. What operand is to select and when is to use is the real of art. Look at the following line code.

/s 3 def.

In above line, def. is used to define top of the dictionary on dictionary stack. S is the name and 3 is the value. Slash ensure that the name s will be pushed onto the stack. Top operand on the stack is value and below the value is key that is name. Operators can also be defined by def. for example,

/pam Aadd mulB def.

The above line of code is definition of an operator that will take two top most operands adds them and multiplies the result with the next operand on stack. The above definition is also called procedure. A very mandatory thing to know about defining procedure is that first operator is invoked and then elements will be evaluated.

Postscript, basically have main purpose to draw graphics on page. One beautiful and pretty thing about Postscript is that it even draw text as graphic. It has "draw and fill mechanism". To do this it has the following sequence.

- **New Path:** Start the path with the newpath operator.
- **Construct :** Construct the path out of line segments and curves.
- **Draw and Fill:** Draw the path with the **stroke operator** or fill it in with the **fill operator**.

This is the basic sequence and we can modify it to do more complicated things. Following is an example to illustrate this:-

```
newpath
    1 inch 1 inch moveto
    2 inch 1 inch lineto
    2 inch 2 inch lineto
    1 inch 2 inch lineto
closepath
stroke
```

3.2.10. Elixir Postscript File

In elixir specific file there are multiple sections. Every section has its own meaning and function. But the area I have to work on is header area and page setup area.

Header area contains the header information like creator, language and version etc. whereas begin page setup area contain the printing option information. Only the options that are mentioned here can be updated in this postscript. Pictorial representation of these two sections is as under:-

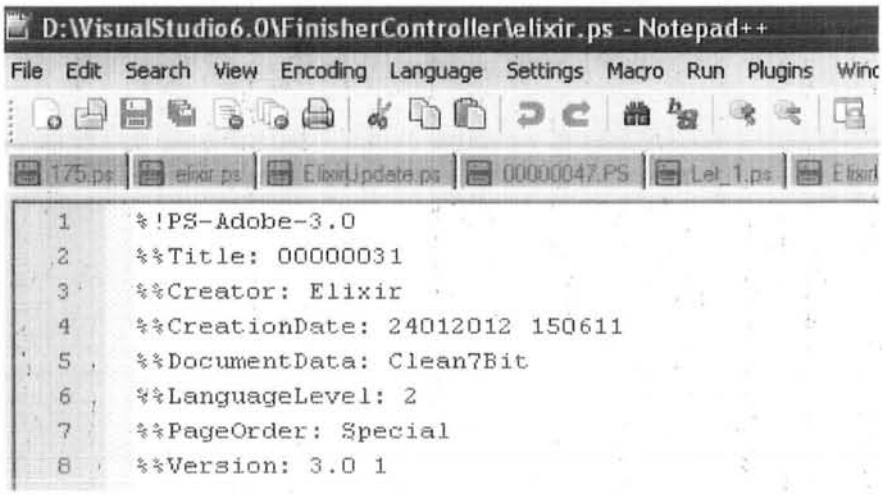


Figure 3.1: Elixir PS File Header

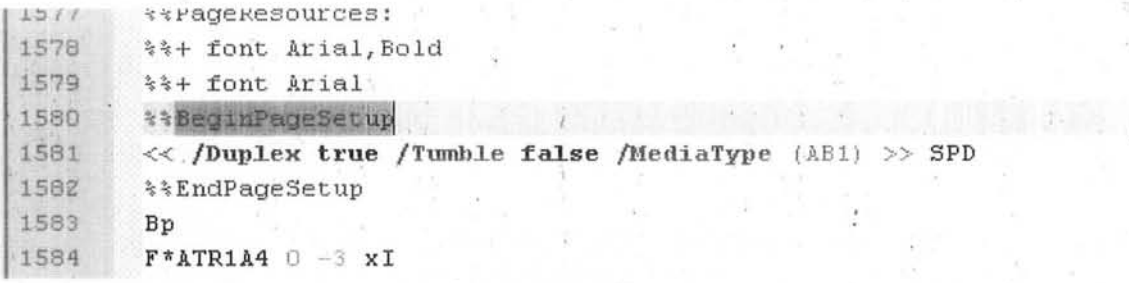


Figure 3.2: Elixir PS File Begin Setup Area

In figure 3.2, within begin page setup area a word SPD exists. It means set page device (SPD). Printing options is selected from PPD and matched here with options shown here before SPD. If selected option is matched then it postscript value is inserted after SPD and options is removed from here otherwise selected option is droped.

3.3. PostScript Printer Description (PPD)

All devices (any output device which PS enabled contains a postscript interpreter) have different features set and there is possibility that devices have same feature set invoke feature in different ways. Each device has postscript printer description file associated with it.

A PPD file is a static representation of the features available on a device. It contains information on the features available on a device as it is shipped from the factory. PPD files are text files that describe an approach to use miscellaneous feature of device. Every device have different features such as memory size, default setting , fonts, page size, finishing features such as duplex printing and stapling etc. All this necessary information about device's features, how to change setting etc. is provided by PPD files. PPD files are stored on the Host computer that is going to use these files. These files are accessible to host computer. Applications parse PPD files and discover the available feature list. Before discovering features blind parsing is done. There exist such structures in PPD file that allow blind parsing. This parsing purpose is to select the device with the help of user interface.

Then application build user interface from list of features extracted from PPD file. To invoke each feature PPD file also has Postscript language code. Now user select feature such as duplex printing or A4 page size, Postscript language code for each option is extracted and placed at appropriate place in the output file before output file is sent to device. Postscript Printer Description (PPD) files are created by vendors to describe the entire set of features and capabilities available for their PostScript printers.

A PPD also contains the Postscript code (commands) used to invoke features for the print job. As such, PPDs function as drivers for all PostScript printers, by providing a unified interface for the printer's capabilities and features. For example, a generic PPD file for all models of xeror Color LaserJet contains:

```
*PPD-Adobe: "4.3"
*% Adobe Systems PostScript(R) Printer Description File
*% Copyright 1993-97 Electronics for Imaging, Inc.
*% All Rights Reserved.
*% Permission is granted for redistribution of this file as
*% long as this copyright notice is intact and the contents
*% of the file is not altered in any way from its original form.
*% End of Copyright statement
*%EFFileVersion: 2.0
*FormatVersion: "4.3"
*FileVersion: "1.0"
*PCFileName: "EFXJX404.PPD"
*LanguageVersion: English
*LanguageEncoding: ISOLatin1
*Product: "(Fiery XJ DocuColor 40)"
*PSVersion: "(2017.103) 1"
*ModelName: "Fiery XJ DocuColor 40 Color Server v2017.103"
*%ShortNickName: "Fiery XJ DocuColor 40 v2017.103"
*ShortNickName: "Xerox DocuColor40 with XJ+4.2"
*NickName: "Fiery XJ DocuColor 40 Color Server v2017.103"
*Manufacturer: "Xerox"
```

Figure 3.3: PPD File Header

This is known as header of PPD file.

3.3.1. Format

The syntax of PPD files is a simple line-oriented format where the options, defaults, and invocation strings (PostScript language code sequences that change a feature setting) are made available through a regular set of keywords.[3]

3.3.2. Terms Used in PPD

Basically three keywords are used in PPD files.

- Main keyword
- Option keyword
- Query keyword

3.3.2.1. Main keyword

This describes the device features such as font size, page size and duplex etc. It has subsets that are **default keyword, Option keyword** and **query keyword** . As name shows default keyword describes default state of device. It is denoted as *Default. Default is always prefixed of default state description. Every main keyword and option key word has a name and value. Syntax of main key word is as below.

*MainKeyword: value

e.g. *OpenUI *EFFinisherOpt/Finisher : PickOne

here "*OpenUI" tells that it is an interface option. "*EFFinisherOpt/Finisher" is name of main key word and "PickOne" is value.

Main keyword has two general classes.

- Informational
- User Interface

Informational: This provides information about a feature, for instance; each font will contain how much memory space? Etc. This information is only beneficial for application and need not to add in user interface.

User Interface keyword: This provides information about those features that are going to appear in user interface. They also provide code to invoke selectable features. User interface entries are represented by structure keyword *OpenUI/*CloseUI.

3.3.2.2. Option Keywords

When there are several choices against a feature then option keywords are provided. That means this feature has multiple attributes. For example, there might be many multiple page sizes listed in the *PageSize section.

*OpenUI *PageSize/Page Size :PickOne

*OrderDependency: 25 AnySetup *PageSize

*DefaultPageSize: A4

*PageSize Tabloid/11x17: "....."

*PageSize Legal/Legal: "....."

*PageSize Letter/Letter: "....."

*PageSize Statement/Statement: "....."

*PageSize A3/A3: "....."

*PageSize B4/B4: "....."

*PageSize A4/A4: "....."

*PageSize B5/B5: "....."

*PageSize A5/A5: "....."

*PageSize Executive/Executive: "....."

*?PageSize: "....."

*CloseUI: *PageSize

3.3.2.3. Query Keyword

This describes the sequence of code. It return the device's state at the time of code downloading to the device. This is used by the application to determine the state of device. It is not necessary that every main keyword also has query keyword. Query keyword is optional and only defined if they are useful and is completely optional otherwise.

There exist statements to describe these keywords. A statement is a single object of main keyword, option and value. There are different formats for statements and every statement in printer PostScript description file is specified by one of these formats.

- *MainKey
- *MainKey: StringValue
- *MainKey: "QuotedValue"
- *MainKey: ^SymbolValue
- *MainKey OptionKey: StringValue
- *MainKey OptionKey: "InvocationValue"
- *MainKey OptionKey: ^SymbolValue

Logically belonging statements are grouped together and is called an **entry.** An entry has several instances of main keyword, *default keyword, and query keyword.

3.3.3. Parsing Main Keywords

Parsing main keyword's detail is following.

• If Main key word does not exists it means feature does not exist on that particular device.

• If a main keyword is not recognized, the entire statement (including multiline code segments) should be skipped. However, the point of the *OpenUI/*CloseUI structures is to allow new main keywords to appear without a print manager explicitly recognizing them. The most functionality will be provided to the user if a print manager handles all main keywords that occur within the *OpenUI/*CloseUI structure, displaying them and invoking their associated code to the best of its ability. Unrecognized main keywords that occur outside of the *OpenUI/*CloseUI structure should be skipped.[4]

• A * in the first column denotes the beginning of a main keyword. Any text or white space before the * should be considered an error.

• The case of main keywords is significant. For example, *PageSize is distinct from *Pagesize. The proliferation of keywords that are the same textually except for case is strongly discouraged.

• Every main keyword can be of maximum length 40 characters.

• Main keywords can contain any printable ASCII characters within the range of decimal 33 to decimal 126 inclusive, excluding colon and slash.

• Delimiters for main keywords are space, tab, colon, or newline. After the initial * symbol is recognized, all characters through (but not including) the next space, tab, colon, or newline character are considered part of the main keyword.

• If a main keyword is not terminated with a colon or newline, an option keyword can be expected.[4]

3.3.4. Parsing Option Keywords

The option keywords of a given main keyword are surrounded by the *OpenUI/*CloseUI keywords. Other things to remember about parsing option keywords:

An option keyword begins with the first character after white space after a main keyword. The case of option keywords is significant. For example, A4 is distinct from a4. Maximum length of an option keyword is 40 characters; it includes any extensions or qualifiers separated by dots.

Option keywords can contain any printable ASCII characters within the range of decimal 33 to decimal 126 inclusive, except for the characters colon and slash, which serve as keyword

delimiters. Once the option keyword is encountered, and before it is properly terminated, a space, tab, or newline character should be regarded as an error.[4]

3.3.5. Comment Statements

PPD file structure also supports comments in the file. Like C or C++, PPD file also have comment structure. "*%" is used to write comments.

3.3.6. PPD File Structure

While we talk about PPD file structure, the first line must be like this:-

*PPD-Adobe: "nnn"

"*PPD-Adobe" indicates that it is a PPD file and "nnn" indicates the version number. It may be like this " *PPD-Adobe: "4.3" ".

This line generally followed by comment line that starts from *% characters. After comments there is header of PPD file. And below header PPD detail exists.

Summary

In this chapter, postscript file and postscript printer description files are explained. General parsing technique for PPD file is described. File structure of PS and PPD files is discussed.

Chapter No. 4

User Interface

4.1. Introduction

User interface is the junction between a user and a computer program. An interface is a set of commands or menus through which a user communicates with a program [5]. There must be an interface to communicate with software to get functionalities/services. Interface can be menu based or command based. In this application, menu based interface is provided.

When we print a document, user can select multiple features, such as number of copies, page size, staple etc. through a user interface such as a print panel or a command line. Providence of these features is based on PPD parsing. It means interface is constructed by parsing PPD file for the selected device. For example, the PPD file contains a list of paper sizes supported by the device. A user interface can display that list to the user and allow the user to select a paper size from the list.

4.2. User Interface and Parts

Main part of user inter face are shown in the following diagram.

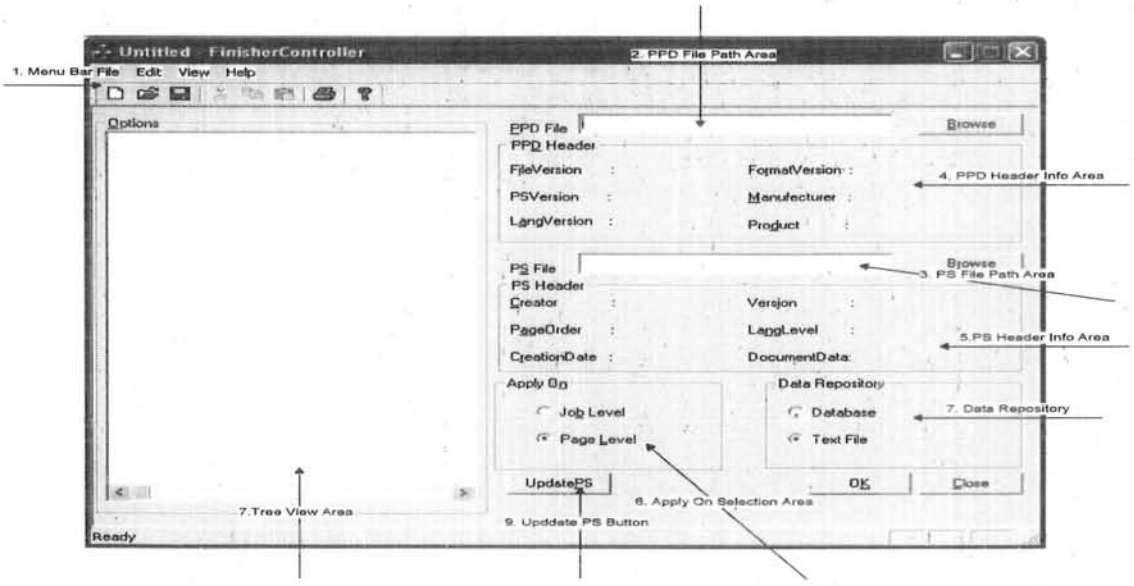


Figure 4.1:User Interface layout

4.2.1. Menu Bar

Menu bar has multiple options. Interface of the system is menu based. User can start or exit application from Start or Exit menu command.

4.2.2. PPD File Path Area

In this area, user can enter input file path. Or user can click browse button to get file path with the help of file dialog.

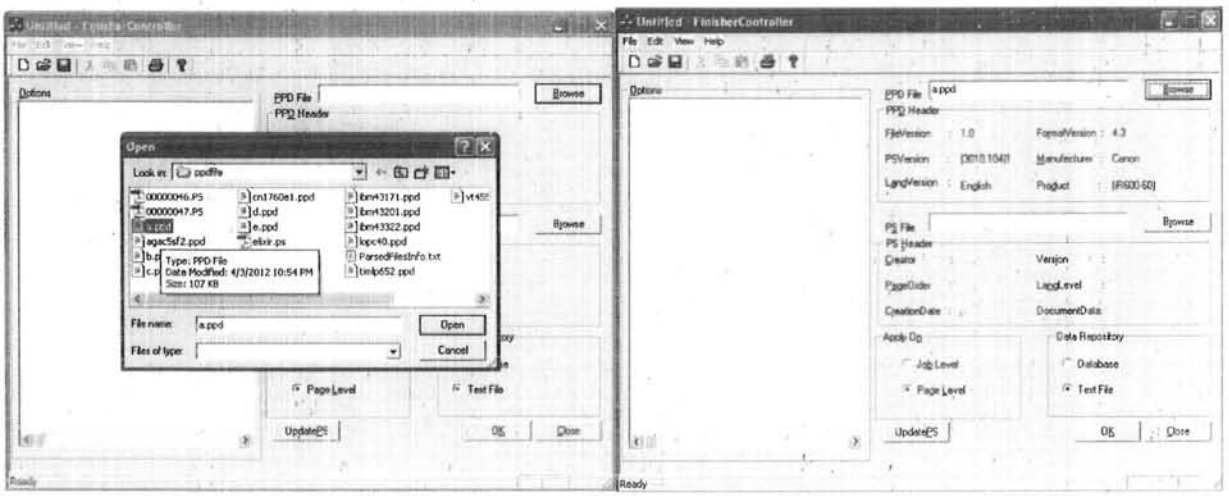


Figure 4.2: PPD file input interface

4.2.3. PS File Path Area

In this area, user can enter input file path. Or user can click browse button to get file path with the help of file dialog.

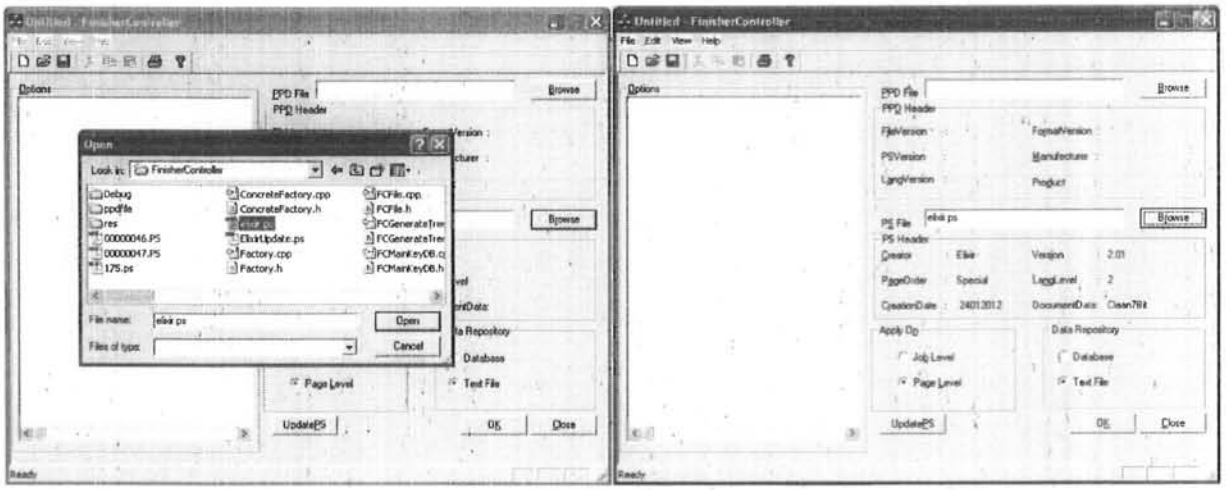


Figure 4.3: PS file input interface

4.2.4. PPD Header Area

This area represent PPD header info. It gives the PPD file information like File Version, Format Version, language version etc.

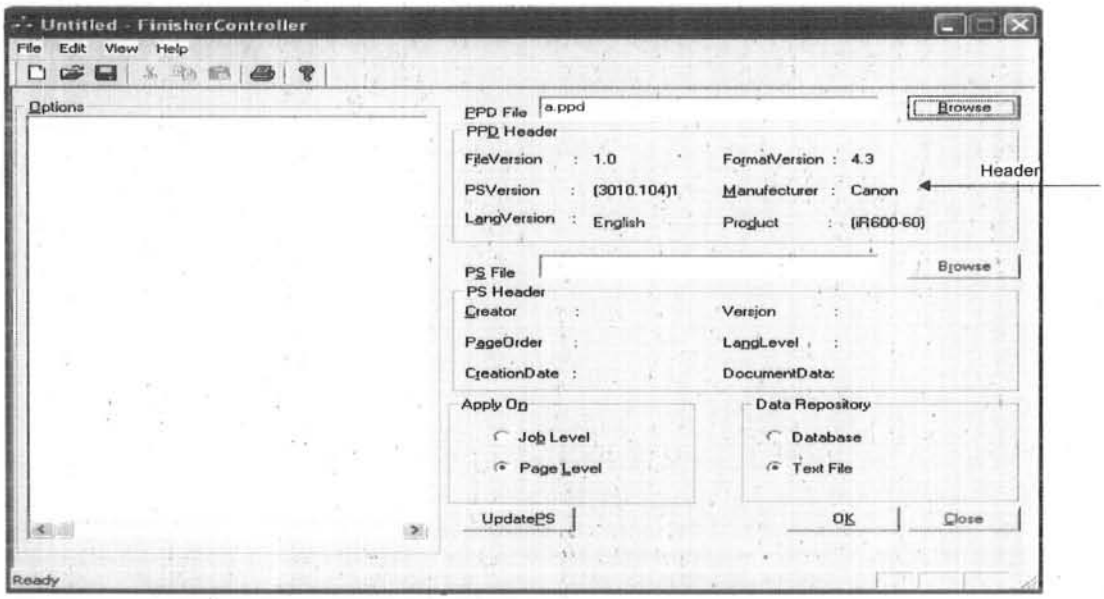


Figure 4.4: PPD header area on user interface (UI)

4.2.5. PS Header Area

This area represent PS header info. It gives the PS file information like creator, page order, language level and version etc.

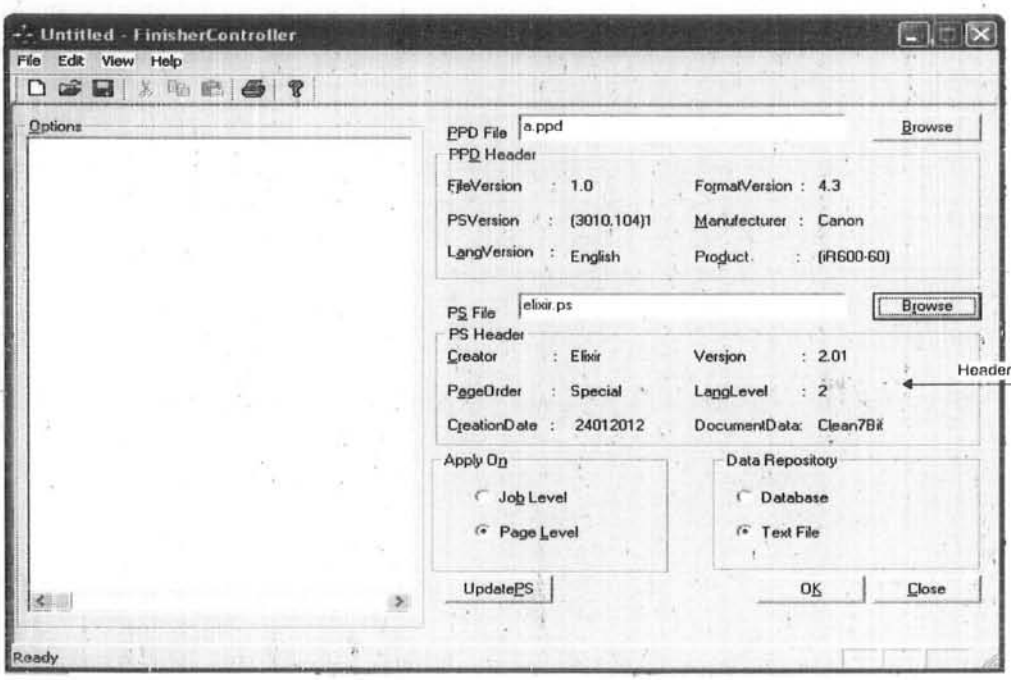


Figure 4.5: PS header area on UI

4.2.6. Apply on Area

This area will allow the user to select apply on area. User can select radio button to select options according to his desire.

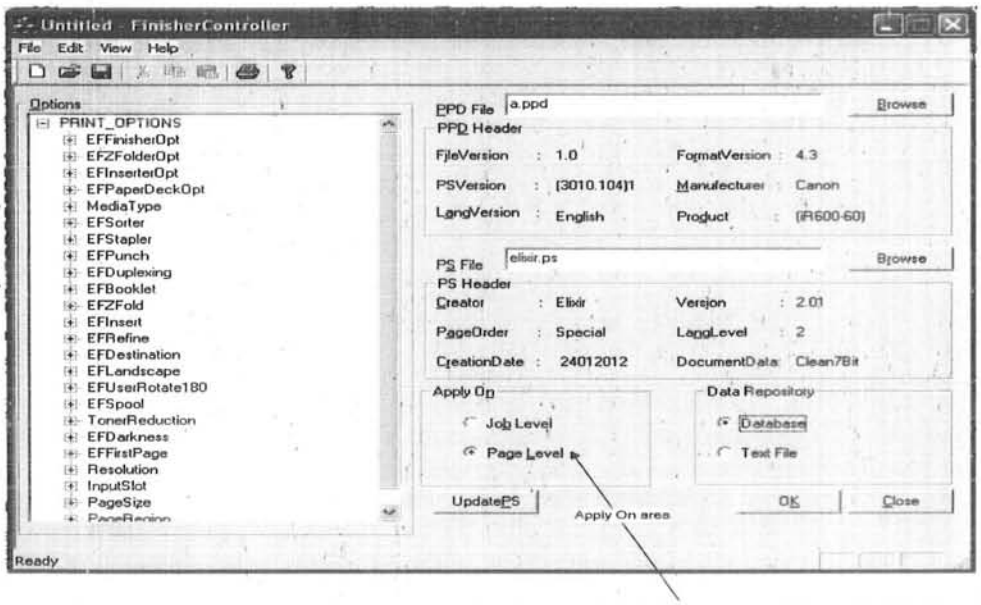


Figure 4.6: Apply on selection area

4.2.7. Tree View

Required Parsed information is saved in tree data structure and displayed in tree view are as shown in snapshot.

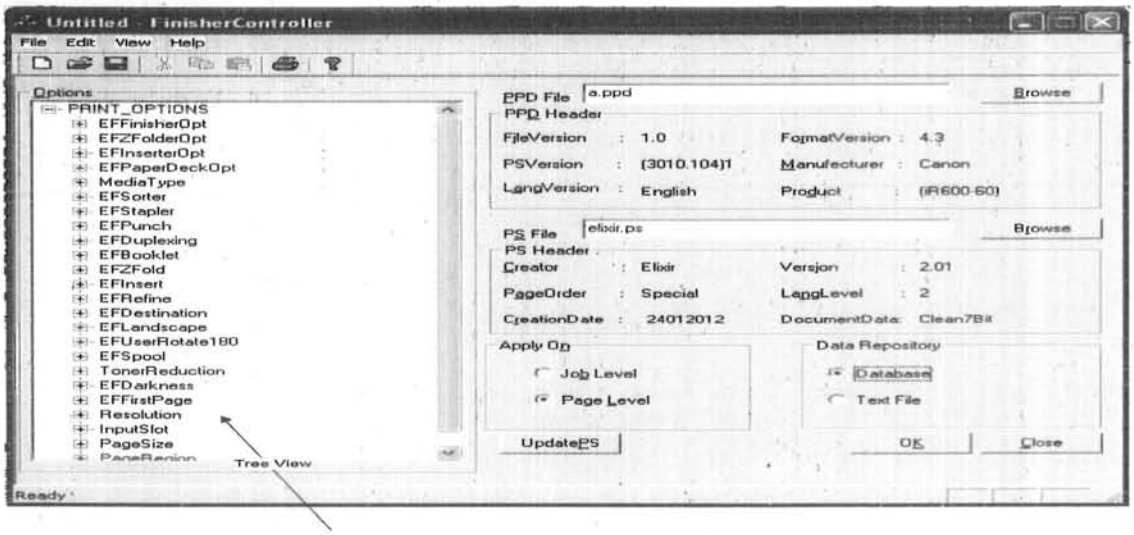


Figure 4.7: Tree view on UI

4.2.8. Data Repository

In this section, user can select data repository for data storage. In this system MS Access database is supported. .

4.2.9. Update PS Button

This area contains a button. System will allow the user to click Update PS button and Postscript file will be updated. After Pressing Update button system will display a message that PS file has updated.

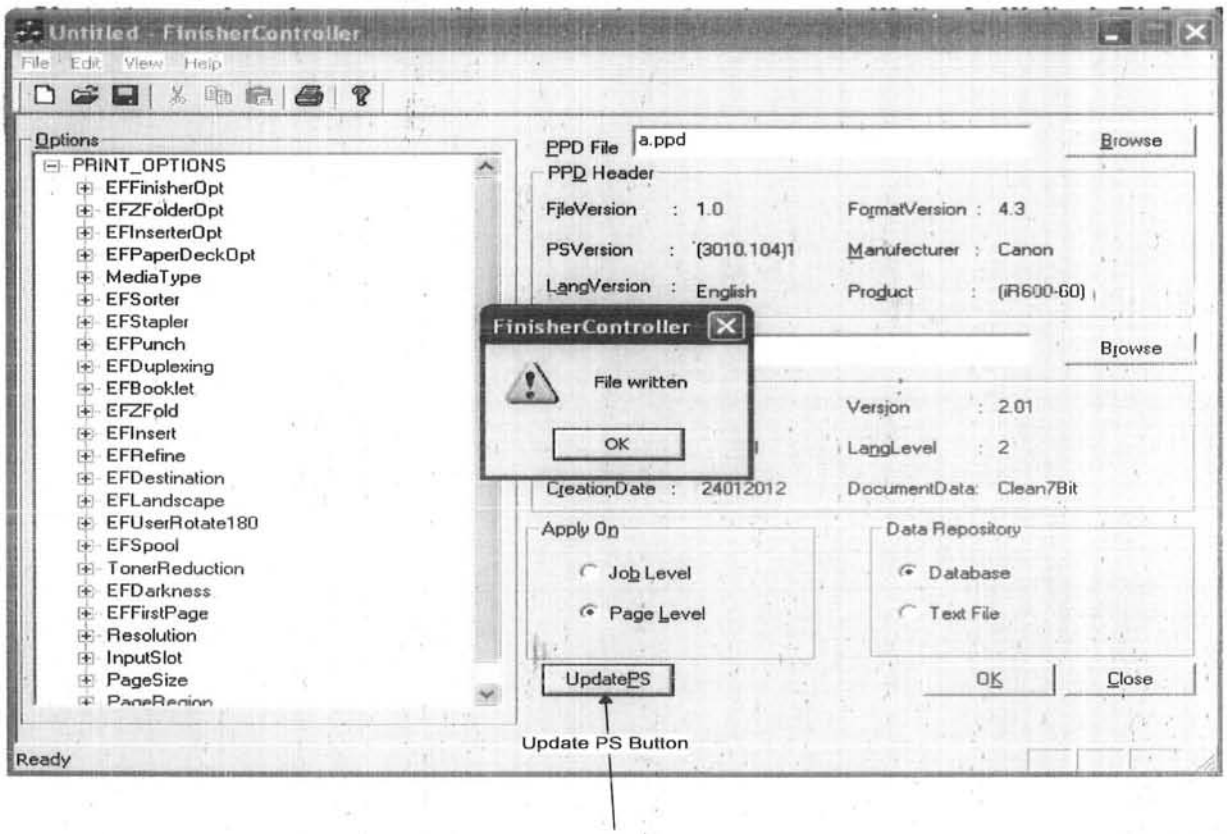


Figure 4.8: PS file Updating button on UI

Summary

In this chapter, User interface is discussed and different areas and sections are described.

Chapter No. 5

System Design

5.1. Design Strategies and Coding Techniques

In this chapter, system design strategies and coding techniques are discussed. In the era of rapid application development, object oriented approach is of great worth. In this section we decide how to design our system and what approaches should follow to design system. We will elaborate coding techniques to implement our system. In next phase, system's architecture and detail design is explained.

5.2.1. Language Selection

I have selected Visual C++ as programming Language with power of STL and MFC library will be used to develop system. The reasons behind choosing this Language and tool are following:

5.2.2. MFC

Object oriented methodology promotes the idea of separation of concern. I want to separate business logic from presentation logic. MFC's Document/View architecture provides MVC architecture to facilitate separation of business logic from presentation logic. In MFC, Document acts as **Model** and view acts as **View/Controller.** Standard Template Library (STL) provides great assistance in coding and streamlines the code. We can use document and view or only view according to our design.

5.2.3. Abstraction

VC++ provides abstraction through classes. That streamlines the code implementation. Inheritance and polymorphism plays a significant role in MFC library so we don't have to mess up with heaps of code.

5.2.4. Documentation

Microsoft provides MSDN, a very well structured documentation to assist developers. A lot of stuff regarding VC++ help also exists on internet and available in books.

5.2. Coding Techniques

For the implementation of this project I will keep following coding guidelines in use:-

- Pascal casing is used for class names.
- Camel casing is used for the function and attribute names.
- First few digits of the attribute names describe attributes type.

5.2.1. Object Oriented Approach

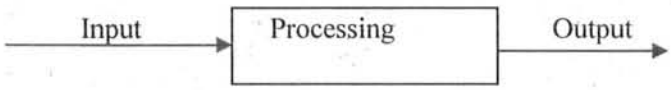
MFC is completely object oriented library. Also our components will be implemented by using Object Oriented approach.

5.2.2. Pattern Oriented approach

Pattern is a general solution of a problem in a specific domain. Design patterns will be used to enjoy the features promised by Object Oriented in reality. Among design patterns, Façade pattern will be used to provide a common interface so to keep code clean, Adapter method will be used to wrap the functionality of MFC classes. Wrapper/adaptor pattern will be used to wrap functionality of MFC classes. [6]

5.3. System Design

A system design is an activity that defines the architecture, interfaces, components and data for the system. A system is a collection of components that work together to realize some objective forms a system. Basically there are three major components in every system, namely input, processing and output [7].



It is the age of Object Oriented methodologies. So Object Oriented Approach is used for system design and implementation. System Design is based on user requirement and analysis of the system. In this chapter two stages are illustrated to represent system design.

- Software Architecture Design
- Software Class Design

5.3.1. Software Architecture Design

In general design, I shall present architecture diagram to show abstract level design. This will give the overview of the system.

5.3.2. Software Class Design

In structural or detail design, I will present *static* and *dynamic* structure of the system. Class diagram will show the static structure and sequence diagram will show dynamic structure of the system.

5.4. System Architecture Diagram

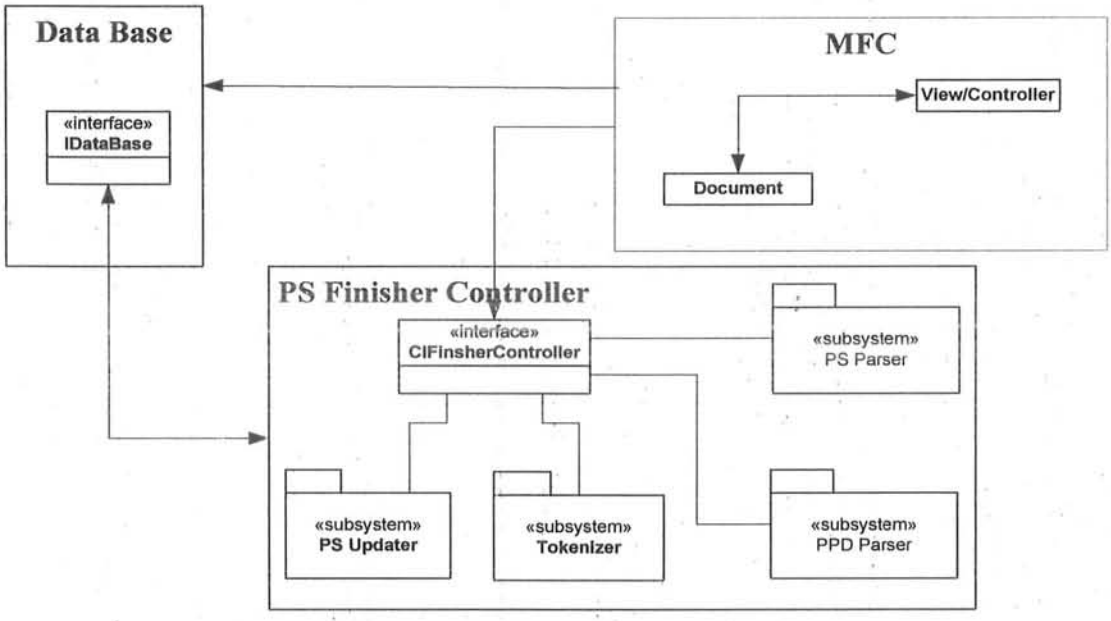


Figure 5.1: System architecture

This diagram illustrate that our system will be implement using MFC framework. System architecture has four main parts that are user, MFC framework, PS Finisher Controller and database.

5.4.1. User

User will interact with the application and also have knowledge about relation between PPD and PS files and the system (Postscript finishing feature controller).The user have the skills and expertise to use application.

5.4.2. MFC Framework

MFC is object oriented wrapper of windows API for C++ provided by Microsoft [8]. It is a collection of classes. CObject is base class for MFC library. Most of the classes are derived from CObject class either directly or indirectly. The MFC Application Wizard makes easy to create an application skeleton with a document class and a view class. Microsoft foundation class (MFC) library supports Multiple views to represent data. I will use MFC interface in my system. It has two parts that are document and view/controller.

4.4.2.1. Document

Document act as a central repository of the application. Document is Model on which all views are dependent. A change propogate through out all application from the document.[9]

4.4.2.2. View/Contrller

View is the visual representation of data. It displays the output of the application.User will interact with application through the view. Application take input from the user through view. Contoller creats an event handller for the application.[10]

5.4.3. PS Finisher Controller

This module contains all processing related to system. Different parts of PS Finisher Controller are CIFinisherController, PPD Customised Parser, PS Customised Pareser, Modifier.

4.4.3.1. CIFinisherController

This class will act like a manager. It manages all classes except MFC library classes. It is like facade class. It provide interface between MFC and rest of the classes of the system.

5.4.3.1. PPD Customized Parser

It is a sub system of the whole system. It will read PPD info and extract required information form the PPD file. It will give the tree view of parsed information. It will let the user to select options according to user's own choice.

5.4.3.2. PS Customized Parser

In this subsystem, system will read PS file and parse the required information form it. It can be document level, job level or page level.

5.4.3.3. PS Updater/Mapper

In this section, PS file will be updated. PS updater will get selected information and search the location in parsed PS file. After matching the place user selected value will be replaced in PS file. Thus PS file will be updated.

5.4.3.4. Data Base

This section will contain **database processing**. It will give interface for database connection. It handles all database processing. PPD parsing information will be stored in database.

5.5. Structural Design/Class Diagrams

A class diagram describes static structure of the system. It is part of unified modeling language. Class diagram shows classes, data, functions and relationship of system and interaction between the system's classes.

5.5.1. MFC Framwork

MFC class structure shows MFC classes hierarchy. MFC classes detail is as under.

5.5.1.1. CObject

CObject provides basic services, including serialization support, run-time class information, object diagnostic output and compatibility with collection classes. The derived classes can have only one CObject base class, and that CObject must be leftmost in the hierarchy.[11]

5.5.1.2. CCmdTarget

The base class for the Microsoft Foundation Class Library message map architecture. Key framework classes derived from CCmdTargetinclude CView, CWinApp, CDocument, CWnd, and CFrameWnd. CCmdTarget includes member functions that handle the display of an hourglass cursor. Display the hourglass cursor when you expect a command to take a noticeable time interval to execute.[12]

5.5.1.3. CDocument

CDocument supports standard operations such as creating a document, loading it, and saving it. The framework manipulates documents using the interface defined by CDocument. Users interact with a document through the CView object(s) associated with it. To implement

documents in a typical application, you must do the following: Derive a class from CDocument for each type of document. Add member variables to store each document's data. Implement member functions for reading and modifying the document's data. The document's views are the most important users of these member functions.[13]

5.5.1.4. CWnd

The CWnd class provides the base functionality of all window classes in the Microsoft Foundation Class Library. A CWnd object is created or destroyed by the CWnd constructor and destructor. The Windows window, on the other hand, is a data structure internal to Windows that is created by a Create member function and destroyed by the CWnd virtual destructor. Within the Microsoft Foundation Class Library, further classes are derived from CWnd to provide specific window types. Many of these classes, including CFrameWnd, CMDIFrameWnd, CView, CDialogand CMDIChildWnd are designed for further derivation [14].

5.5.1.5. CFrameWnd

To create a useful frame window for your application, derive a class from CFrameWnd. There are three ways to construct a frame window.

- Directly construct it using Create.
- Directly construct it using Load Frame.
- Indirectly construct it using a document template. [15].

5.5.1.6. CMainFrame

A class derived from CFrameWnd. CMainFrame class is related to the PS Finisher Controller application. The CMainFrame utility class presented here helps to implement a modal dialog's behavior for any CFrameWnd derived window class in an easy way [16].

5.5.1.7. CTreeCtrl

A "tree view control" is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Here Each item consists of a Main key and list of sub items i.e. main key and option keys [17].

5.5.1.8. CView

A view is attached to a document and acts as an intermediary between the document and the user. A view is a child of a frame window. More than one view can share a frame window, as in the case of a splitter window. The relationship between a view class, a frame window class, and a document class is established by a CDocTemplate object [18].

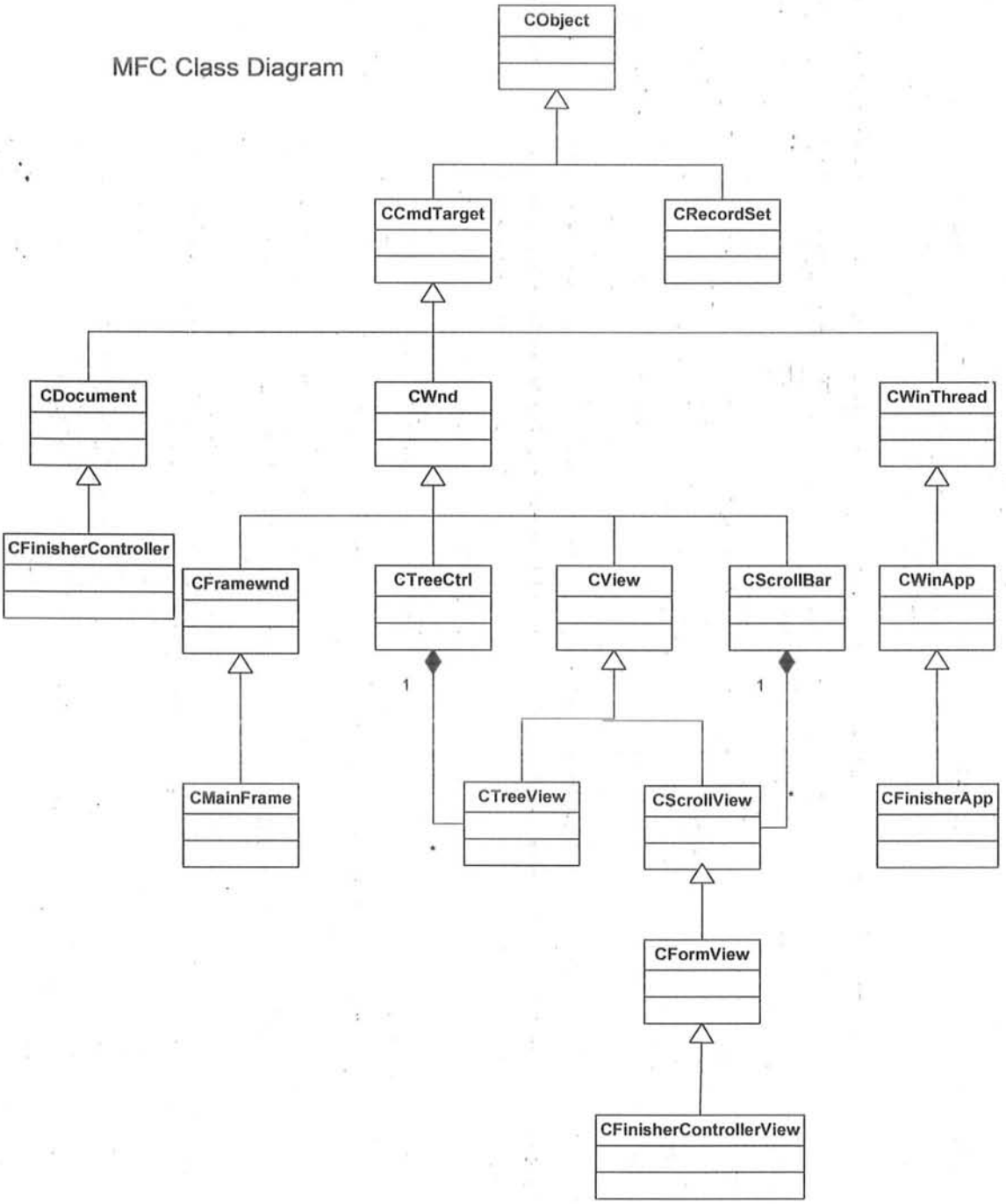


Figure 5.2: MFC class diagram

5.5.1.11. CFinisherController

CFinisherController behaves like a façade for all classes of the system to interact them with MFC form. CFinisherController class communicates all complex subsystem classes with MFC form.

5.5.2. System Structure

Finisher controller has different parts/components that will communicate each other to perform a task. Components and their interaction is described in next paragraphs. Main parts of system structure are Filing, Parser, Tokenizer, Database Manager, Tree View Manager, Updater.

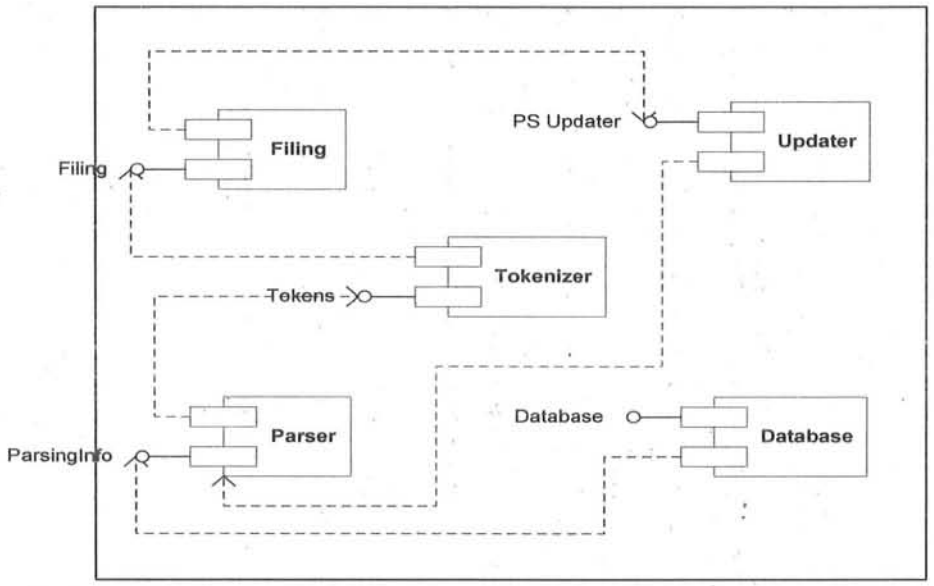


Figure 5.3: Component interaction

A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, methods and the relationships between the classes [19]. Classes of each part and their responsibilities are elaborated in the following sections.

5.5.2.1. Filing

In this section file is handled, PPD and PS file handling is done through this mechanism. There are an interface class for filing and an implementation class. Furthermore there are two specialized implementation classes on for handling PS file and other is for PPD file.

a). CFCFinisherController

CFCFinshingController is the class that manages all classes except MFC classes. This class has an attitude like a facade class in facade pattern. CFCFinsherController class creates the objects of each subclass. This class provides a mechanism by which all sub-classes collaborate with each other. CFCFinsherController class makes sub-classes compatible to collaborate with server class by giving a generic interface. This class shows the concept of object oriented by encapsulation and information hiding. Due to this class the MFC structure have no idea about the application subclasses.

b). IFCFile

IFCFile class is an interface class for file system. It has all member function declarations. It has pure virtual functions. It gives interface for file processing either it may be a Postscript printer description (PPD) file or Postscript (PS) file.

c). CFCFile

CFCFile class is a general class that is derived from interface class. It has implementation of file handling mechanism. It gives read and writes mechanism for Postscript file.

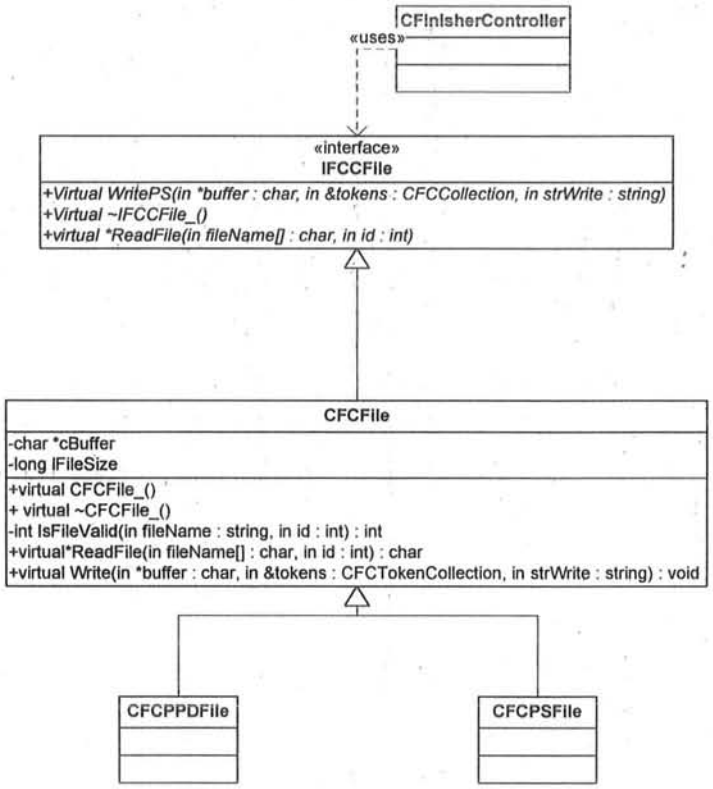


Figure 5.4: Filing structure

5.5.2.2.Tokenizer

a). CFCToken

CFCTokens class store the tokens of each individual PPD file or PS file after the tokenizing from the CFCTokenizer class. CFCTokens class consists token string, token ID and line number.

b). CFCTokenCollections

CFCTokenCollections class is a collection class and store the complete information of tokens related to complete PPD File or PS file. CFCTokenCollections class is inherited from the CCollections class.

c). CFCCollections

CFCCollections class is the base class of all classes which have collection PPD or PS file. CFCCollections class has the common functionality which is common in all collection classes.

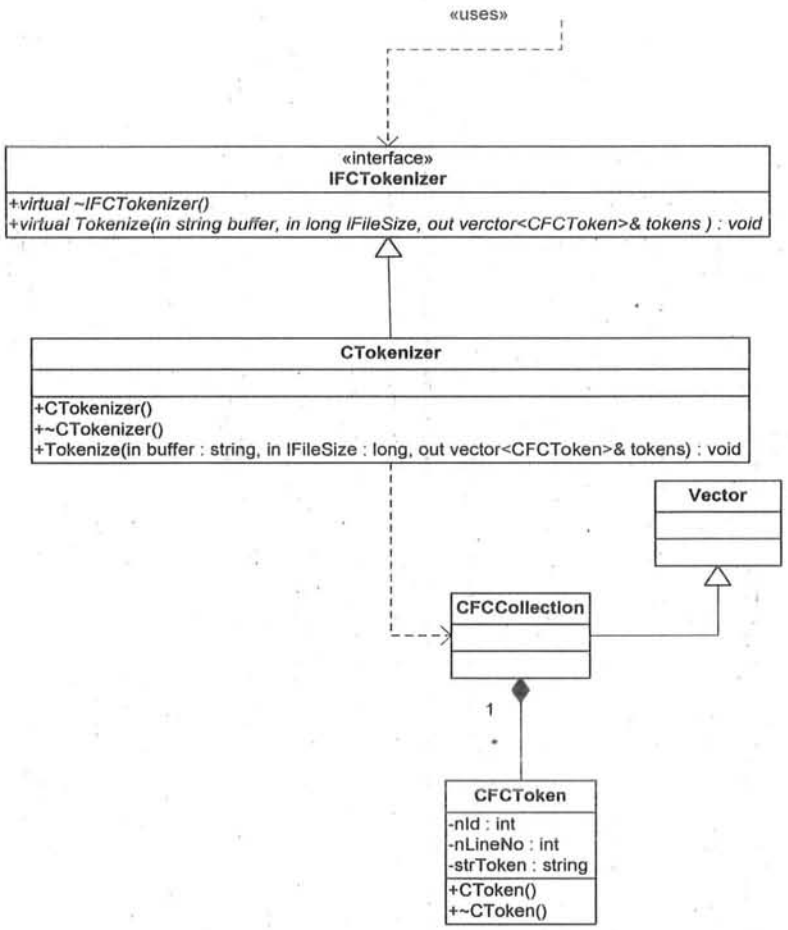


Figure 5.5: Tokenizer structure

d). CFCTokenizer

CFCTokenizer class is an implementation class that takes input and tokenizes the file into tokens. This class takes the PPD file or PS file and split them into tokens. CFCTokenizer class give the information of each token in the form of token string, line number and token ID.

e). IFCTokenizer

IFCTokenizer class is an interface class. That provides interface for tokenizer to Finisher Controller.

5.5.2.3. Database

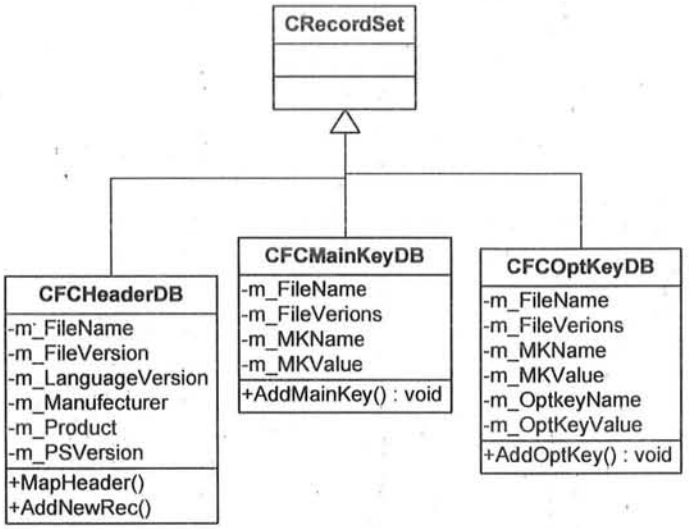


Figure 5.6: Database classes structure

b). CRecordSet

this is an MFC class that is provides support for database.

c). CFCHeaderBD , CFCMainKeyDB and CFCOptionKeyDB

These are classes that will handle data by fetching or storing from database. CFCHeaderBD has file information like file name and header information.and CFCMainKeyDB and CFCOptionKeyDB classes have mainkey and option key information.

5.5.2.4. Parser

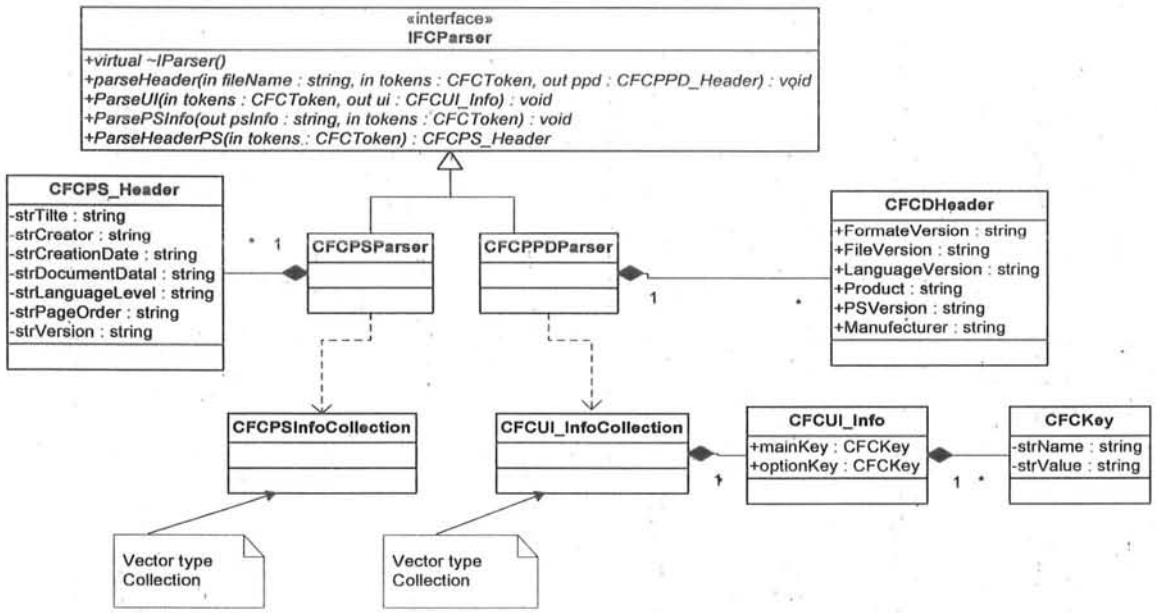


Figure 5.71: Parser structure

a). IFCParser

IFCParser class is an interface class of parser. It will let the subclasses to create the object. It gives common interface for both PPD parser and PS parser.

b). CFCPPDParser

CFCPPDParser class get token collection from CFCTokenizer class. It parses header information and feature information form the PPD token collection. CFCPPDParser class will check database and if file has already parsed then it will not parse it again. If it is not already parsed CFCPPDParser will parse required information and store it into database.

c). CFCPSParser

CFCPSParser class get token collection from CFCTokenizer class. It parses finishing option information and feature information form the PPD token collection. It derives from parser class.

d). CFCKey

CFCKey class will contain information that is strName and strVlaue. Every key has to parts one is Name and other is value.

e). CFCPPD_Info

CFCPPD_Info class has header information for individual file. CFCPPDParser class will parse header information and store it into PPD_Info.

f). CFCUI_InfoCollection

CFCUI_InfoCollection class will contain information for individual PPD file. It is collection of those options that will be displayed on user interface. CFCPPDParser class will parse finishing options from PPD file store it into CFUI_InfoCollection.

f). CFCPSInfoCollection

This class a collection that contains parsed PS file information. Its reference is given to the CFCPSParser.

5.5.2.5. Tree View Manger

a). CFCGeneralTree

CFCGeneralTree *class* will store main keys and option keys extracted from PPD parser. Main key will be parent and option keys will be child. Option keys are collection. This information is stored in CGeneralTree and it will map into MFC tree view for visual representation.

b). CFCTreeNode

CFCTreeNode class is a node of the tree. It is nested in CFCGeneralTree. It holds information of node like node item, node parent and node children.

5.5.2.6. Updater

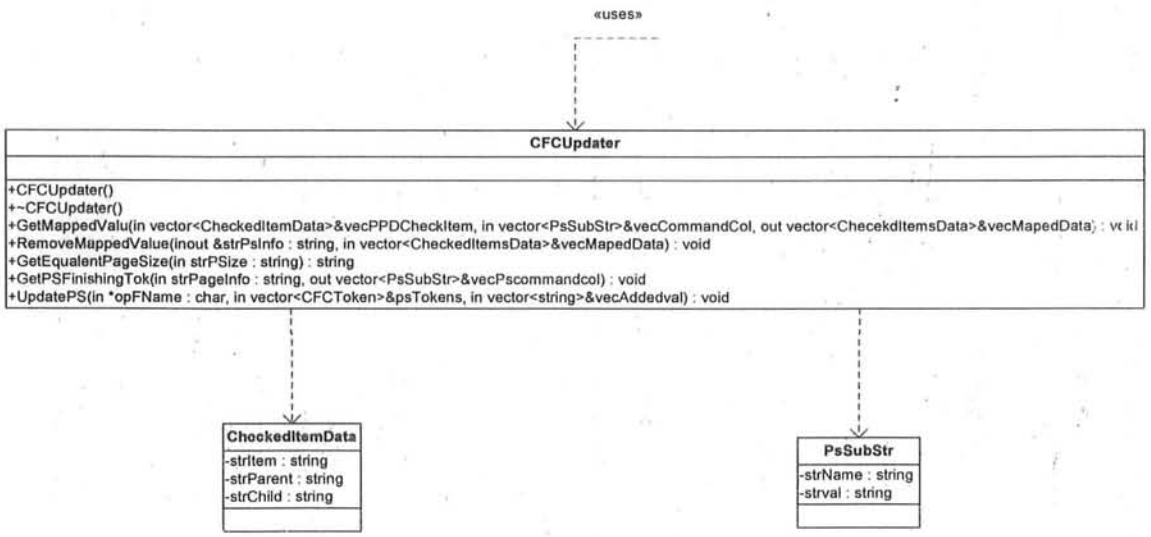


Figure 5.8: PS Updating mechanism

a). CFCUpdater

This is implementation class for updater. All function's implementation held here. This class is inherited from interface class. It is an example of interface inheritance.

5.5.2.7.Finisher Controller Factory

Factory design pattern will also implemented in development of application. There are multiple objects created in the application. Factory will make object creation at same place.it will simplify the design.

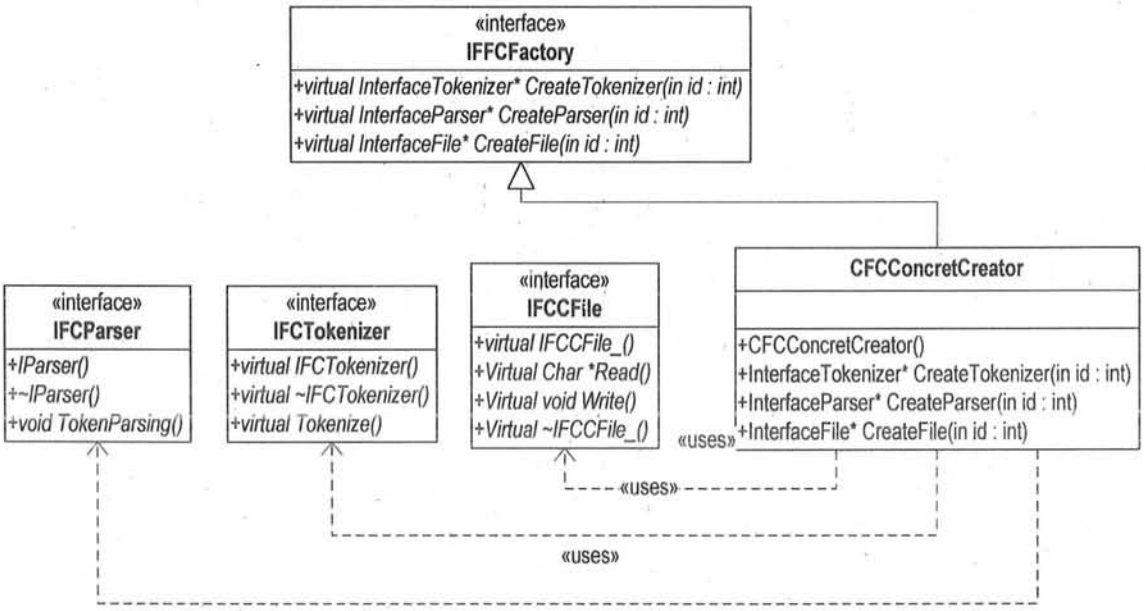


Figure 5.9: Factory Design

5.5.2.8. CFCFinisher Controller (work as Façade)

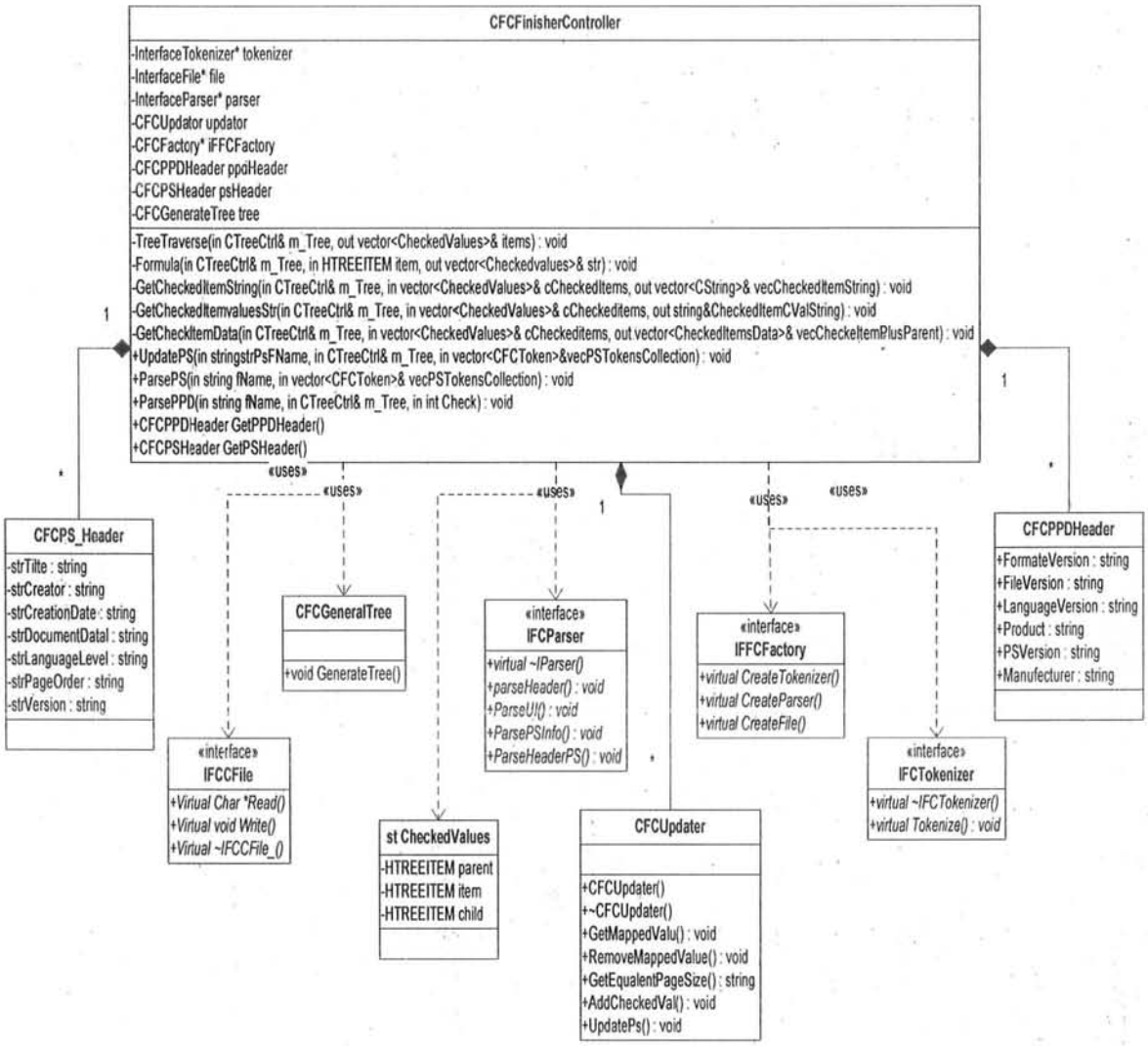


Figure 5.10: CFCFinisher Controller (Façade)

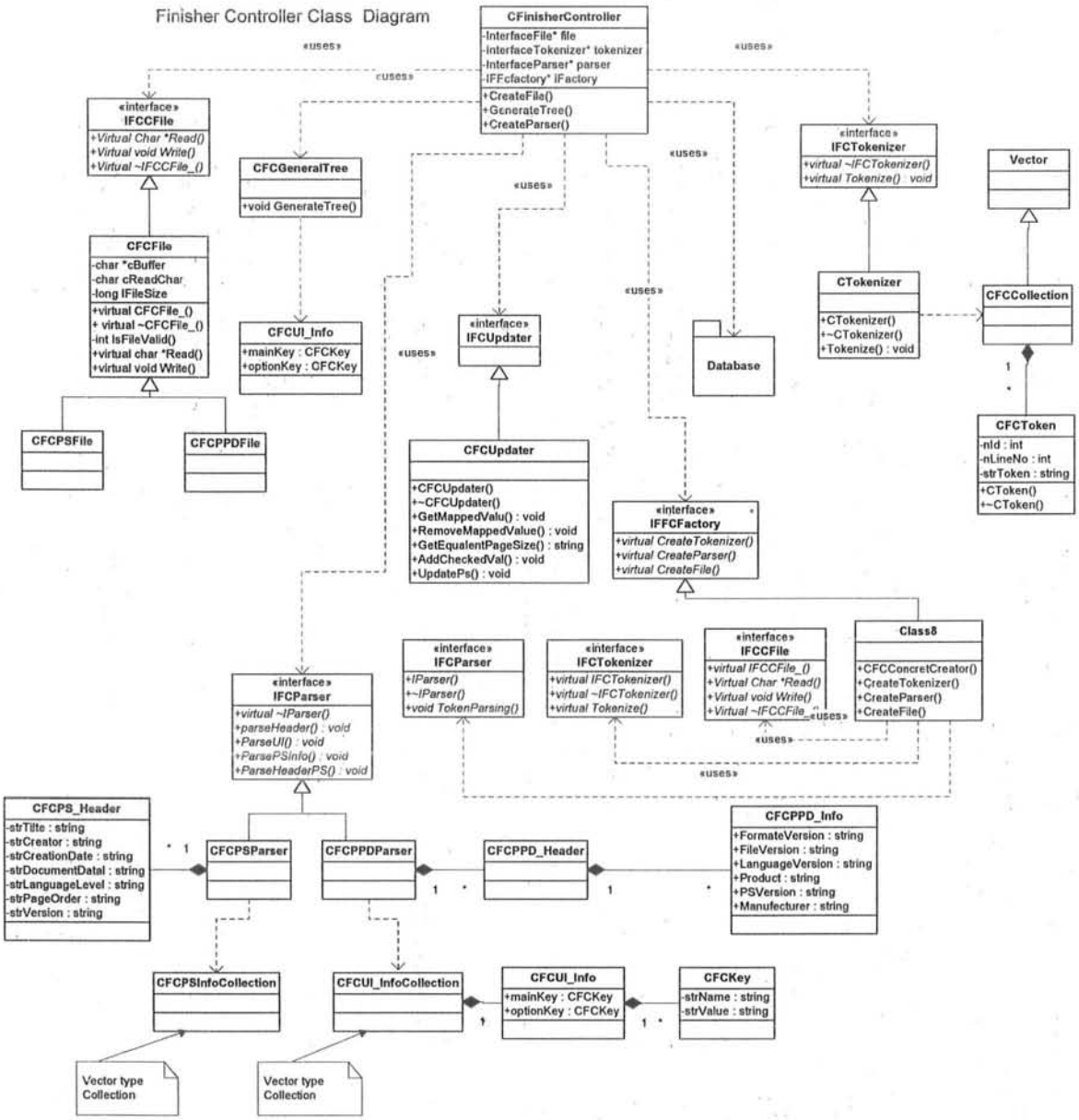


Figure 5.11: Complete Finisher Controller class diagram

Note: In above detail class diagram methods parameters are not shown and component interaction is taken place in CFCFinisherController.

5.6. Sequence Diagram/Dynamic Structure

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows interaction of objects and order in which they collaborate with one another. It is a construct of a message sequence chart [20].

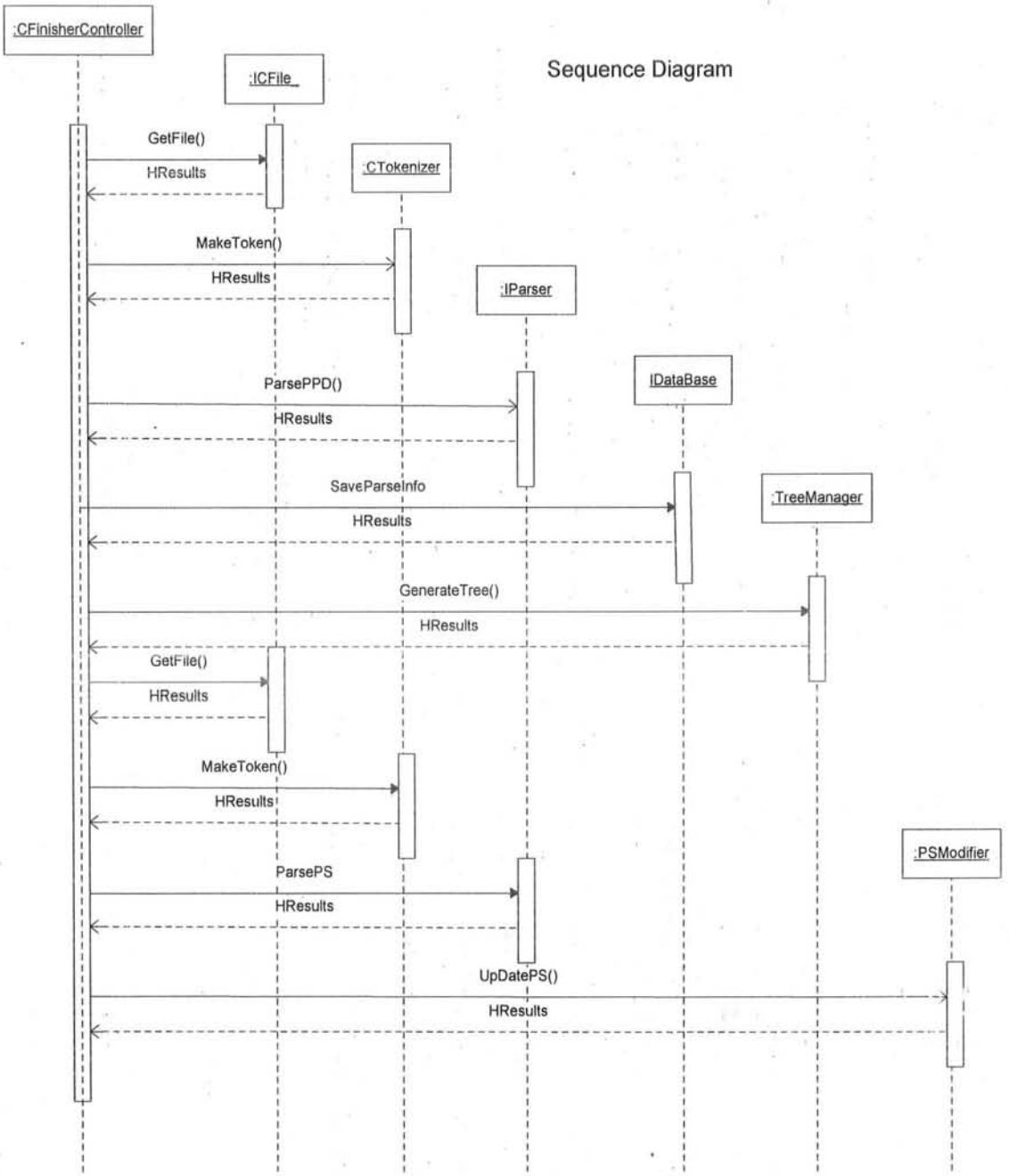


Figure 5.12: Sequence diagram

5.7. Data Repository

Data Repository is a logical (and sometimes physical) partitioning of data where multiple databases which apply to specific applications or sets of applications reside [21]. The term Repository is commonly used to refer a place for storing and maintain data safely.

In this system, data can be store in Text file or Database. MS Access will be used as database tool.

5.7.1. Database

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses [22] .

5.7.2. E-R Diagram

An entity-relationship (ER) diagram is a specialized graphic that illustrates the relationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes [23].

System has following entities.

- File
- MainKey
- OptionKey

5.7.3.1. File

File is an entity that will be stored in database. It has following characteristic. Fileverion, FileName, PSVersion, Product, LanguageVersion, Manufecturer.

5.7.3.2. MainKey

Main Key in another entity with ID, name and value characteristic.

5.7.3.3. OptionKey

It is sub key of a main key.one main key have many sub keys. Every sub key has an id, name and value.

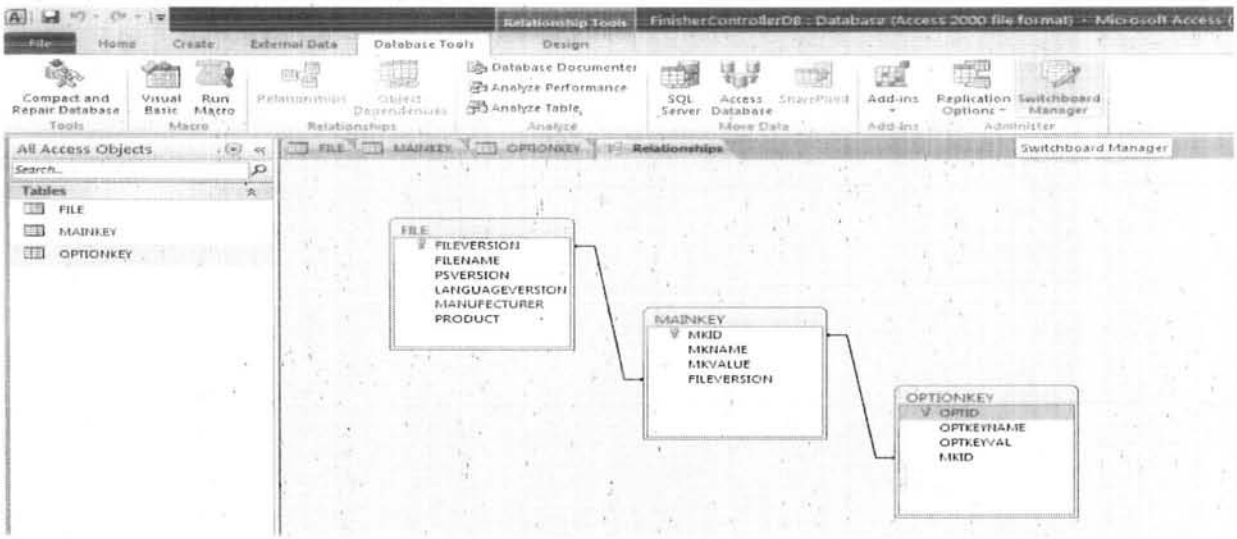


Figure 5.13: E-R diagram

5.7.3. Database Tables

MS Access database tables are show below for the system.

Table 1: File table diagram

FILE | MAINKEY | OPTKEY

FILEVERSION	FILENAME	PSVERSION	LANGUAGEV	MANUFECTL	PRODUCT	Click to Add
1.0	a.ppd	(3010.104)1	English	Canon	(iR600-60)	
1.0	agac5sf2.ppd	(2013.108)9307	English		(AGFAAccuSet:	
1.0	b.ppd	(3010.104)1	English	Canon	(GP605-605P)	
1.0	c.ppd	(3010.104)1	English	Canon	(iR330-400)	
1.0	cn1760e1.ppd	(3010.104)116	English	Canon	(CanonLBP-176	
1.0	e.ppd	(3010.103)9507	English	Epson	(EPL-N2700)	
1.0	timlp652.ppd	(2014.104)16	English		(microLaserPrc	

Table 2: MainKey table diagram

FILE | MAINKEY | OPTKEY

MKNAME	MKVALUE	FILEVERSION	FILENAME
TonerReductio	Boolean	1.0	a.ppd
EFSpool	Boolean	1.0	a.ppd
EFUserRotate1	PickOne	1.0	a.ppd
EFDestination	PickOne	1.0	a.ppd
EFInsert	Boolean	1.0	a.ppd

Table 3: OptionKey table diagram

FILE | MAINKEY | OPTKEY

OPTKEYNAM	OPTKEYVALUE	MKNAME	FILENAME
LLeft	/XJXsetstapler where { pop 8 XJXsetstapler } if	EFStapler	a.ppd
PUpper	/XJXsetstapler where { pop 7 XJXsetstapler } if	EFStapler	a.ppd
PUpperRight	/XJXsetstapler where { pop 2 XJXsetstapler } if	EFStapler	a.ppd
LRight	/XJXsetstapler where { pop 7 XJXsetstapler } if	EFStapler	a.ppd
LUpperLeft	/XJXsetstapler where { pop 3 XJXsetstapler } if	EFStapler	a.ppd
PRight	/XJXsetstapler where { pop 6 XJXsetstapler } if	EFStapler	a.ppd
False	/EFUserRotate180 where { pop 0 EFUserRotate180 } if	EFUserRotate1	a.ppd
True	/EFUserRotate180 where { pop 1 EFUserRotate180 } if	EFUserRotate1	a.ppd
EFUserRotate1		EFUserRotate1	a.ppd
True	/XJXsetzfold where { pop 1 XJXsetzfold } if	EFZFold	a.ppd
False	/XJXsetzfold where { pop 0 XJXsetzfold } if	EFZFold	a.ppd
True		EFZFolderOpt	a.ppd
False		EFZFolderOpt	a.ppd
Tray2	/XJXsettraysel where { pop 2 XJXsettraysel } if	InputSlot	a.ppd
Tray3	/XJXsettraysel where { pop 3 XJXsettraysel } if	InputSlot	a.ppd
Tray4	/XJXsettraysel where { pop 4 XJXsettraysel } if	InputSlot	a.ppd
ManualFeed	/XJXsettraysel where { pop 1 neg XJXsettraysel } if	InputSlot	a.ppd
Tray1	/XJXsettraysel where { pop 1 XJXsettraysel } if	InputSlot	a.ppd
Tray5	/XJXsettraysel where { pop 5 XJXsettraysel } if	InputSlot	a.ppd
AutoSelect	/XJXsettraysel where { pop 0 XJXsettraysel } if	InputSlot	a.ppd
P_Interleaved	/XJXsetmediatype where { pop 3 XJXsetmediatype } if	MediaType	a.ppd
Transparent	/XJXsetmediatype where { pop 1 XJXsetmediatype } if	MediaType	a.ppd
(AB1)	/XJXsetmediatype where { pop 0 XJXsetmediatype } if	MediaType	a.ppd
Interleaved	/XJXsetmediatype where { pop 2 XJXsetmediatype } if	MediaType	a.ppd
Plain	/XJXsetmediatype where { pop 0 XJXsetmediatype } if	MediaType	a.ppd
Letter	/XJXsetpagesize where { pop (Letter) XJXsetpagesize } if	PageRegion	a.ppd
Legal	/XJXsetpagesize where { pop (Legal) XJXsetpagesize } if	PageRegion	a.ppd

Summary

This chapter briefly discusses the design of the system. The System Architecture diagram illustrates that our system will be implemented using MVC architecture. Then describe the structure of a system by showing the system's classes and relationships between them using class diagram. Further, in this section, database for the system is discussed. E-R diagram shows entities and their relationships. Relational database tables for the system are also shown.

In this chapter we have discussed reasons of selection VC++ as development language, design and coding guidelines and techniques. Pattern oriented and object oriented approaches are elaborated for this system.

Chapter No. 6

System Testing

6.1. Introduction

The systematic test is an inevitable part of the verification and validation process for software. Testing is aimed at finding errors in the test object and giving confidence in its correct behavior by executing the test object with selected input values [24]. Software testing is a process that continues throughout the development process. **Test cases are based on use cases**. Testing is a V&V (verification and validation) practice. We verify system functionality according to our requirements and check feasibility of the system. There are three basic **goals** of software testing.

- Error detection. Is output of test matches to the expected result?
- Verification. Are we developing the software/product right?
- Validation. Are we developing the right software/product?

6.2. Testing Techniques

Currently, there are many testing strategies that are in use. System can be tested manually or automatically. I have used two testing techniques for my application testing.

- White Box Testing.
- Black Box Testing.

6.2.1. White Box Testing

White box testing is also known as structural and glass testing. Is this strategy, internal mechanism or code of the system/component is tested. I have used **unit testing** in account of white box testing.

6.2.2. Black Box Testing

Black box testing is also famous as functional testing. It is not related to internal mechanism of system or component. It solely about outputs that are produced in result of given/select inputs

and execution conditions. I have used functional and system testing in account of black box testing.

6.3. Test Cases

A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement (IEEE Standard 610 (1990) [25]. So, a test case is a document that describes the procedure to test the system. My system test cases details are following.

Test Case ID	**1**
Test Case Name	PPD Input Test
Test Category	Black Box
Test Case Object	Input File
Test Role(Actor)	My self
Precondition	System must ready to test/use
Execute	1. Press browse button to get file. 2. File dialog open. 3. First Choose PPD file OR any other than PPD file 4. Click open or press enter
Success Criteria	Only PPD file should be input successfully. System should discard all other non PPD files.
Post Condition	1. System should indicate error message "Invalid file" in case of wrong file selection or no selection. 2. System should show no error in case of PPD file selection.
Result Value	Successful

Test Case ID	**2**
Test Case Name	Tree View Test
Test Category	Black Box
Test Case Object	Tree

Test Role(Actor)	My Self
Precondition	1. PPD file is input successfully 2. Input PPD file must have parsed.
Execute	1. Tree is generated successfully after input the PPD file 2. Expand tree by clicking nodes(parent and child) 3. Drag child node and drop on parent node.
Success Criteria	Tree should expand properly. Drag drop must not allowed
Post Condition	Tree generation and expansion should be done properly.
Result Value	successful

Test Case ID	3
Test Case Name	Select Options Test
Test Category	Black Box
Test Case Object	Tree
Test Role(Actor)	My Self
Precondition	1. Application must be in active state. 2. Application must have input PPD file. 3. Input PPD file must have parsed. 4. Tree must be generated.
Execute	1. Expand the Tree. 2. Tree has default values selected. 3. Select/deselect values by checking/unchecking check box.
Success Criteria	Options should be selected according to main key values
Post Condition	Options selected successfully.
Result Value	Successful

Test Case ID	4
Test Case Name	PS input test
Test Category	Black Box

Test Case Object	Input PS File
Test Role(Actor)	My self
Precondition	System must be in active state
Execute	1. Press browse button to get file. 2. File dialog open. 3. First Choose PS file OR other than PS file 4. Click open button or double click on file 5. PS file is parsed and header info must displayed on screen
Success Criteria	Only PS file should be input successfully. System should discard all other non PPD files.
Post Condition	1. System should indicate error message "Invalid file" in case of wrong file selection or no selection. 2. System should show no error in case of PS file selection.
Result Value	Successful

Test Case ID	5
Test Case Name	PS Updating test
Test Category	Black Box
Test Case Object	UpdatePS button
Test Role(Actor)	My self
Precondition	1. System must ready to test/use 2. User has given an input PPD file 3. PPD file has parsed 4. Tree View has generated 5. Option has selected 6. User has given Input PS file to update 7. Apply on setting has done
Execute	1. Press button to update file 2. Selected options and PS parse info is mapped
Success Criteria	Mapped selected options must be updated in PS file

Post Condition	1. System must show "PS file updated" message after successful update 2. System must show error message in case of failure
Result Value	successful

Test Case ID	**6**
Test Case Name	Exit application test
Test Category	Black Box
Test Case Object	EXIT button
Test Role(Actor)	My self
Precondition	1. System must be in active state
Execute	1. Press EXIT button
Success Criteria	System must be exit
Post Condition	System close successfully
Result Value	successful

Summary

In this chapter, testing techniques used for system testing are described and test cases for this system are enlisted.

Reference:-

[1] http://www.requirementsauthority.com/functional-and-non-functional.html

[2] Bittner, Kurt & Spence, Ian (2003). *Use case modeling*

[3] PostScript Printer Description File Format SpecificationVersion 4.3

[4] PostScript Printer Description File Format SpecificationVersion 4.3

[5] http://www.webopedia.com/TERM/U/user_interface.html

[6] Larman, C., "Applying UML And Patterns", 2nd Edition

[7] http://www.nos.org/htm/sad1.htm

[8] http://en.wikipedia.org/wiki/Microsoft_Foundation_Class_Library

[9] http://msdn.microsoft.com/en-us/library/4x1xy43a%28v=vs.80%29.aspx

[10] http://developer.qt.nokia.com/doc/qt-4.8/model-view-programming.html

[11] http://www.informit.com/library/content.aspx?b=Visual_C_PlusPlus&seqNum=36

[12] http://msdn.microsoft.com/en-us/library/x9w7txst%28v=vs.80%29.aspx

[13] http://msdn.microsoft.com/en-us/library/e59dtf8h%28v=vs.80%29.aspx

[14] http://msdn.microsoft.com/en-us/library/1xb05f0h%28v=vs.71%29.aspx

[15] http://msdn.microsoft.com/en-us/library/za93adby%28v=vs.80%29.aspx

[16] http://www.extrabit.com/fractalviewer/srcdocs/classCMainFrame.html

[17] http://www.cppdoc.com/example/mfc/classdoc/MFC/CTreeCtrl.html

[18] http://msdn.microsoft.com/en-us/library/ezc3635w%28v=vs.80%29.aspx

[19] http://en.wikipedia.org/wiki/Class_diagram

[20] http://en.wikipedia.org/wiki/Sequence diagram

[21] http://www.learn.geekinterview.com/data-warehouse/dw-basics/what-is-data-repository.html

[22] http://searchsqlserver.techtarget.com/definition/database

[23] http://databases.about.com/cs/specificproducts/g/er.htm

[24] http://www.update-it.com/documents/eurostar1993.pdf

[25] http://www.kaner.com/pdfs/GoodTest.pdf