DEDICATION

Dedicated to our parents whose love and care have made us possible to reach this position, dedicated to our teachers whose hard work and knowledge have made us possible to develop a practical application and dedicated to our sisters ,brothers and friends whose sincerity and friendship will provide us with shelter and security.

DECLARATION

This is declared that this project Form Builder For Document Management System was solely prepared by Muhammed Junaid Arshed, Syed Muhammad Khaliq-ur-Rahman Raazi and Wasim Imran. All source code written by them and documentation prepared for Form Builder was their individual and group efforts and not a single part of the project was copied from any other source. The three members of group and National University of Sciences and Technology (NUST) have equal rights of using this software and source code and no one else is allowed to use the software or any part of the code without the permission of all members.

Muhammed Junaid Arshed(Syndicate Leader)

_____

Syed Muhammad Khaliq-ur-rahman Raazi

_____

Wasim Imran

_____

Brig.Dr.Muhammad Akbar(Project Supervisor)

_____

Table of Contents

Abstract

Form Builder for Document Management System is a MDI application that is used to design forms. In Form Builder, documents are forms that are designed in this software.

The main specialty in this project is its compatibility with printing variable data.

Presently, Form Builder does not provide printing solution but they can be incorporated in future as Form Builder is totally compatible with printing. Form Builder uses Flat Files as database tool that gives compatibility to work in future in PPML ,a XML technology used to provide variable printing solution .Also its specialty is its functionality of saving the design in SVG format which is also a XML technology that gives platform independence,B2B communication and makes possible the integrity of Form Builder with printing solution as language used for printing is PPML which is also a XML technology, so two technologies of XML can be easily integrated. Also its functionality is its excellent and user friendly user interface which makes the software easy to use for any new user.

About Project Report

The project report for “Form Builder for Document Management System” is comprised of

- Introduction to core concepts of Form Building and XML
- Software Requirements Specifications
- Object Oriented Analysis of the project
- Future extensions

The report consists of various sections. Section 1 contains an overview and introduction to the core concepts involved in the project. Sections 2 and 3 cover Project and Software requirements specifications. Object Oriented Analysis is covered from section 4 to section 11. The project report is concluded in section 12, covering the achievements and future extensions.

Section 1

INTRODUCTION

1.1 Overview

Documentation is one of the most key processes during the product development in industry. It is considered to be one of the vital components delivered with the product. When it comes to Information Technology industry, the documentation is of pivotal importance and software engineers are trained to produce and update the documents regarding software projects. Keeping the importance of documents in view, some software giants took the initiative to develop software products which handle a large amount of data very efficiently. Some of the prominent names were Xerox, Adobe, Corel, Elixir and Microsoft. Following the footsteps of these key organizations in industry, a large number of software organizations are now working in the field of data handling and printing and a large number of form designers and editors are being developed in the organizations across the globe.

Besides documentation, there is also a need for handling and saving variable data. This variable data is placed on the different documents and then printed. Some examples of the variable data documents are National Identity Cards, Passports, Bills, Employee records etc. The variable data is usually in excess of Tera bytes or Pita bytes. Handling and printing this variable data is one of the hottest research topics in the industry and data companies are developing the more smarter solutions to win the race. They have developed their own formats and print streams. For example IBM has developed its own format of print stream known as AFP, Xerox has developed its print stream format known as VIPP, Elixir has developed its format known as elix. As far as saving of variable data is concerned a lot of research is going on in this area. The goal of the researchers is to develop a universal format for serialization of data. So that the data can be viewed ,edited and printed on any platform and this data should be acceptable by the large amount of text editors, form designers and print streams. In pursuit of the above mentioned goals, US Department of Defense developed a format known as Standard Generalized Markup Language(S.G.M.L).This format was used to save the data in the complex data structures so that the data can be accessed and retrieved more efficiently. As the name suggests, SGML is a markup language and markup with

the data is in the form of tags. These tags store the information about the type of the data and some of the tags also contain the information about the presentation of the data.

Later, in the field of data handling, a very important breakthrough came in 1993, when Hyper Text Markup Language (HTML) emerged on the scene. Hyper Text Markup Language was based on the concepts of SGML. It was developed to display data on the Web. Development of HTML became the basis of the Internet revolution. In the beginning, only 5000 web pages were written in HTML and now in 2002, the number of HTML are in the excess of billions.HTML pages are viewed in special viewing software known as Web browser.HTML solved the problem of handling of data and porting of data on different platforms, to a large extent but it has some shortcomings. The deficiency in HTML is that it is a presentation oriented language. It takes care of presentation format of the data but it does not offer any solution for keeping track of semantics of the data. What data in conveying, how data is structured semantically, how the hierarchy of semantics is developing are some of the important issues. Keeping in view the shortcoming of HTML, Extensible Markup Language (XML) is developed. XML is yet a evolving language. It is a very powerful language for handling data.It contains technologies both for semantics and presentation of data. XML is also developed on the concepts of SGML. In HTML, tags are predefined and there is a standard set of tags which can be incorporated in the documents but in XML, the tags are defined by the developer. This feature of XML provides a great deal of flexibility to the developer as he is not limited by a predefined set of tags. XML is now widely accepted by a large number of text editors, form designers and browsers so it is becoming a universal format.

Keeping in view the increasing importance of variable data handling solutions, we decided to undertake a project by ELIXIR Technologies which is FORM DESGINER for Office Documentation System. The form designer can design and draw like classical form designers and it can perform any sort of text editing as well. The most exciting feature of this form designer is that it can save the data in XML format. To be more precise, it saves data in a new and revolutionary XML based format which is known as

Scalable Vector Graphics (SVG) format. Scalable vector graphics not only presents the core functionality of the XML but it also provides the high resolution rendering of the graphics objects. Incorporation of SVG certainly makes this product a very useful product.

1.2 Document Scope

This is the final documentation which will be delivered with the product. It contains *Project Specifications, Descriptions of different phenomena, Details of Interfaces, various kinds of UML Diagrams, References, Bibliography and Appendices*. Precisely, the documentation will be comprised of:

1. Project Specification
2. Software Requirements Specification
3. Use Cases
4. Sequence Diagrams
5. PDOL
6. Class List
7. CRC
8. CRD
9. Test Cases
10. Problems during Implementation
11. Future Extensions

1.3 Basic Functionality

Form Builder for Document Management System is a software for designing and printing of forms. The forms designed by the Form Builder are used to print with the variable data. This variable data is very large in volume. The size of the variable data may be in the excess of Tera bytes or Pita bytes. It usually resides on a main frame computer. This variable data is downloaded in a simple text file after removing the metadata associated with it. The text file in which the variable data is downloaded is called as “Flat File”. The size of the flat file may be in excess of Giga bytes or Tera bytes. After the creation of flat file, layout of the form is prepared. The variable data from the flat file is then associated with different portions of the form layout. The Form Builder creates two kinds of trees. One is created for the management of the form layout and other is created for the management of flat file. The titles from the flat file can be dragged and dropped onto the form layout to associate different portions of form layout with the variable data. After this step, the form is sent to RAM of the printer for printing. The printer takes the incoming data stream, associate it with the corresponding portion of the form layout and then print the form.

1.4 Problems with the Existing Form Designers

The existing form designers pose some serious and considerable problems. A major problem is that they are confined to a specific format i.e. the form layout can be saved and opened in a specific format of a particular form designer. This format usually belongs to particular vendor and therefore supported by a limited range of products offered by that vendor. This format is not supported by form designers and graphics rendering tools from the other vendors.

Another problem with the existing form designers is that they are confined to a specific platform. Their format can be used in particular platform and it is not properly visible in other platforms. Moreover format doesn’t visible uniformly when viewed in different environments and platforms.

The solution to both of the above mentioned problems is a development of standardized format which could be accepted and implemented by a large number of vendors. SVG can be one such format. It is XML based format. Since XML is platform independent so SVG is platform independent as well. The Form layouts saved in SVG can be viewed uniformly across different platforms and since SVG is accepted by a large number of vendors as a standard format so it is supported by a large number of tools.

For the above mentioned reasons, we have used SVG as a storing format of the Form Layouts.

1.5 Overall Description

1.5.1 Project Specification

1. The Form Builder is a Win32 application.

2. XML version 1.0 is supported by the Form Builder.

3. Form Builder is to be developed on Microsoft Visual C++ version

6.0.

4. XML and SVG used in the Form Builder is parsed by the Microsoft

XML Parser version 4.0.

5. Integrate Form Designer and data layout.

6. Handle huge volumes of variable data.

7. Create customized layouts of the documents.

8. Data definer mechanism (It is done with the help of a grid control).

1.5.2 Project Description

The function of the Form Designer for the Office Management System is to design various kinds of forms used in different kinds of applications.

The Form Builder basically consists of following four modules :

1. Designer
2. Converter
3. Translator
4. Grid Control

Designer is used to design the layout of the form. It contains a “Designer Area” where the user draws the layout of the form. Designer module also contains different kinds of design tools and basic shapes like circle, square, line, rectangle and it also facilitates free hand drawing. Drawing is done with the help of mouse and very elementary users are able to draw very sophisticated designs.

Converter is used to save the layout of the form and then it converts the layout into Scalable Vector Graphics format. Converter actually generates the SVG code for the layout, at the backend. This code confirms to W3C standards and specifications of SVG.W3C is a regulatory body for the standardization of various web technologies, so generating SVG code according to the original specifications of W3C will make the code re-useable and interoperable on any graphics rendering tool which supports SVG.

Translator is used to translate the format saved in SVG format back into the applications format, so that it can be editable in design area. It parses the SVG code and then displays the last saved layout onto the design area.

Grid Control is a very user friendly module incorporated in the FORM BUILDER. The variable data is positioned onto the prepared form layout in different applications. This

variable data is downloaded from a server or Main Frame and saved in a special kind of text file called as “Flat File”. The flat file is analogues to a Relational Database but flat file neither contains meta data nor it stores the data in a organized and readable manner. Due to poor readability, it is virtually impossible for a user to sort out and organize the records and fields manually. Grid Control offers a very user friendly solution to this problem. Flat file is loaded into the grid and the data is organized in the columns and alternate rows of the grid. In the empty rows, a user can give the title of the fields and after giving the titles the user can drag these titles to a “Grid Tree” which maintains the proper classification of the variable data with the help of the user defined titles. These titles are then dragged onto the different portions of the form layout in design area to associate the variable data to these portions. After associating the variable data to different portions of the form layout, the form is now ready for the variable printing of a huge amount of variable data.

The GUI of the Form designer is designed to facilitate the usage of the software to a great ease. Users with a very little knowledge of computer can browse through the different options with a minimum of efforts and subsequently use the options, drawing tools and grid control.

1.6 Where to Use Form Builders

Form Builder is a vastly used print solution. It is used in

1. Large Enterprises, for printing different kinds of records
2. Software Organizations, for generating complex structured documentation.
3. Air Craft and Automobile industry, for printing complex and richly structured user manuals
4. Education Institutions, for printing different kinds of application forms and records.
5. Financial Institutions, for printing different kind of sophisticated and complex structured documents.
6. In government institutions, for designing and Printing Identity cards.

7. In Service industry, for printing bills and vouchers.
8. Printing utility bills.

1.7 Software Project Constraints

1. Form Builder for Document Management System supports Scalable Vector Graphics (SVG) format for the serialization of the data. SVG is supported in Internet Explorer 5.5 and above and NetScape 6.0 and above, so the designed forms will not be viewed, if opened in the IE versions, less than 5.5.
2. With the help of SVG, we can implement animation support in our online form layouts. But this feature is not implemented due to shortage of time.
3. The Form Builder is developed on Win32 platform , it is only tested in Windows environment. It is not tested on other platforms.
4. XML version 1.0 is supported. If the designed forms are edited with XML version 2.0 parsers then the parsers will not able to render them correctly.

Section 2

PROJECT SPECIFICATIONS

2.1 Statement of Purpose

Form Builder for Office Management System is a XML based form designer. This software is used to design different kinds of forms. The form layouts are designed with the help of different kinds of design tools. These forms are then saved in SVG format. SVG is a cutting edge and sophisticated XML format which is platform independent and it is supported by a large number of Graphics Rendering tools and software vendors. Saving the form in SVG will certainly make the forms editable in a large number of Graphics tools which support SVG. Moreover, using the SVG format will make the forms visible uniformly across different platforms and help to enhance the product scope to large number of tools and platforms.

2.2 Scope of the Product

FORM BUILDER for Office Management System is form designer to design and print complex structured forms. It offers various kinds of drawing tools and a “Design Area”. The drawing tools include basic shapes like line, different kinds of rectangles, squares and circles. It also offers support for free hand drawing. Form Builder offers three kinds of window. The first is “Tree” window. The second is “Designer” window and the third one is “Grid Control” window. The Tree window has two kinds of Tree views. The first tree view is called as “Designer View” and the other view is called as “Grid View”. The second kind of window is called as “Designer” window. It offers design area to the user where he can design the layout of the form. The third window is “Grid Control” window. This window presents a grid for opening of a flat file. User designs the layout of the form in Designer window with the help of different kinds of drawing tools. The drawn objects in the designer window are maintained in the Designer Tree View in Tree window. The tree view of the drawn objects makes the navigation easy through the form objects. The “Grid View” maintains the tree view of the objects present in the grid. The switching between the two different views in the Tree view window is done with the help of Tabs.

The input to the Form Builder is comprised of the designed layout in the designer area and the variable data, which is in the form of flat file, opened in the Grid Control window. Titles are given to the fields of flat file in Grid Control and the Grid Tree View is then populated with the help of these user defined titles in the Grid Control. These populated titles are then dragged and dropped onto the designer area to associate different portions of the designed form layout with them. This layout is then saved in SVG format and it is ready to be printed or viewed in any graphics rendering tool which supports SVG format.

Section 3

SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

Purpose of SRS

The SRS for the Form Builder for Document Management System is written to tell the users what this software is supposed to do. The Software Requirements Specification (SRS) is produced as part of the requirements analysis phase. It presents a complete description of the external behavior of the Form Builder for Document Management System. It is serving the following purposes:

- Communication among users, analysts, and designers,
- Supports system-testing activities, and
- Controls the evolution of the system.

SRS Document Overview

Introduction about the system is given in the first chapter that what is the scope of the product and what is the purpose of this document.

In the second chapter, general description of the product is given which includes its perspectives, functionality, user characteristics, general constraints, assumptions and dependencies.

In the next section of this document, requirements are specified such as functional requirements and external interface requirements. In addition, user, hardware, software interfaces along with non-functional requirements are given.

In the end, possible product evolution is given which shows how the product can be evolved in the future to meet the changing requirements of the people.

3.1. INTRODUCTION

3.1.1 System Statement of Scope

Our purpose is to develop a form builder which will create forms with the help of different kinds of drawing tools. The layout of the forms and the variable data attached to the various fields of layout are stored in Scalable Vector Graphics format which is a new and revolutionary graphics format based on XML. It has following main functions:

1. Creates the layout of a form.
2. Supports the opening and closing of a Flat file which is the main source of variable data.
3. Designs the layout of the forms with the help of different drawing tools and shapes such as rectangle, circle, polygon, lines and free hand drawing tool.
4. Serializes and retrieves the form layouts in the Scalable Vector Graphics format which is a XML based graphics format.

3.1.1.1 Goals and Objectives

Form Builder for Document Management System is a software project based on a very rational concept of XML. This project is assigned to us by Elixir Technologies which is a California based company specializing in print solutions. Its major partners are IBM, Adobe, Oce and Xerox. Elixir provides print solutions mostly in the form of AppBuilders and Form Builders. This project is actually the basis for the development of Elixir's new XML based Form Builder. When we planned this project we had the following goals and objectives at the back of our minds:

- To create Forms which are saved and retrieved in XML format.
- To provide a basis for the development of Elixir's new XML based Form Builder.
- To explore the amazing concepts of Extensible Markup Language (XML) along with its cutting edge technologies.

- To learn the advanced concepts of Microsoft Visual C++ 6.0 by implementing the project in this language.
- To learn Microsoft XML Parser for developing XML applications on Win32 platform.

3.1.2 System Context

Form Builder for Document Management system is actually a project sponsored by Elixir Technologies. The project will act as a basis for the development of their new XML based Form Builder. This project will be further enhanced and will become the part of their Office Management Suite.

3.2. GENERAL DESCRIPTION

This section of the SRS will describe the general factors that affect the system and its requirements. The purpose of this chapter is not to present detailed requirements, but rather to present material that will make the specific requirements easier to understand.

3.2.1 System Functionality

The functionality of the system can be summarized in following steps:

1. The system will be used to create form layouts.

2. The system will be able to open a flat file in a grid control.

3. The system will be able to assign the titles to fields of the flat file.

4. The system will be able to map the assigned titles of the flat file to the various

sections of the form layout.

5. The system will be able to serialize and retrieve the designed form and form layouts

in Scalable Vector Graphics (SVG) format.

3.2.2 User Constraints

The users of this system are solution providers which provide print and document solutions to various small and medium scale enterprises. This system can be used in

1. Large Enterprises, for printing different kinds of records
2. Software Organizations, for generating complex structured documentation.
3. Air Craft and Automobile industry, for printing complex and richly structured user manuals
4. Education Institutions, for printing different kinds of application forms and records.
5. Financial Institutions, for printing different kind of sophisticated and complex structured documents.
6. In government institutions, for designing and Printing Identity cards.
7. In Service industry, for printing bills and vouchers.
8. Printing utility bills.

The users of the system should be aware of:

1. Concepts of XML

The user should have at least a vague idea of XML that how data is structured into its tree structures

2. Concept of Flat File

The user should know about the concept of Flat File. What flat file is? How it is formed and how it is maintained? These are the question whose answers should be known to users

3. Mapping of Flat file

The user should be aware of mapping flat file titles into the designer area.

4. Navigation with the help of Tree structures

As tree structures are an important part of GUI of the system so user should about the navigation with the help of the tree structures.

3.2.3 General Constraints

1. Form Builder for Document Management System supports Scalable Vector Graphics (SVG) format for the serialization of the data. SVG is supported in Internet Explorer 5.5 and above and NetScape 6.0 and above, so the designed forms will not be viewed, if opened in the IE versions, less than 5.5.
2. With the help of SVG, we can implement animation support in our online form layouts. But this feature will not be implemented due to shortage of time.
3. The Form Builder is developed on Win32 platform , it is only tested in Windows environment. It is not tested on other platforms.
4. XML version 1.0 is supported. If the designed forms are edited with XML version 2.0 parsers then the parsers will not able to render them correctly.

3.2.4 Assumptions and Dependencies

Following assumptions are made during the preparation of the project plan:

1. The software is developed for the Win32 platform.

2. The forms can be opened in Internet Explorer 5.5 and above and NetScape Navigator 6.0 and above.
3. XML parser 1.0 is installed on the user's system.
4. The user has an awareness to the concept of flat files.

3.3. SPECIFIC REQUIREMENTS

This section of the SRS will contain a complete description of the external behavior of the software system. It will contain sufficient information and detail necessary to create a design for the system.

3.3.1 Functional Requirements

Form Builder for the Document Management System must be capable of :

1. Designing the form layouts
2. Supporting a grid control
3. Supporting the flat files
4. Serializing the forms in a custom format i.e. fbd
5. Serializing the forms in Scalable Vector Graphics
6. Format (SVG)
7. Opening the custom saved (fbd) forms
8. Opening the SVG forms

3.3.2 External Interface Requirements

3.3.2.1 User Interfaces

Form Builder for Document Management System has a very sophisticated and pleasant user interface. The interface is very easy to navigate even for an elementary user. Different kinds of graphical interface components like tree, tab controls and docking windows are incorporated. The user interface of the system is described as follows:

MENUS

1. File

New	Opens a new file
Open	Opens an existing file
Open Grid	Opens an existing grid
Save Grid	Saves an existing or new grid
Close	Closes the file
Save	Saves the file
Save As	Saves in SVG or fbd format
Print	Prints the file
Print Preview	Shows the print preview of file
Print Setup	Enters into the printer setup
Exit	Exits the application

2. Edit

Undo	Undo the last operation
Cut	Cuts the selected segment of the object
Copy	Copies the objects
Paste	Pastes the copied objects
Clear All	Deletes all objects from designer area

3. View

Tool Bar	Shows the main tool bar
Status Bar	Shows the status bar
Shape Toolbar	Shows the shape toolbar

4. Tools

Line	Draws a line
Circle	Draws a circle
Rectangle	Draws a rectangle
Pencil	Draws free hand
Filled Rectangle	Draws a filled rectangle with selected color
Filled Circle	Draws a filled circle with selected color
Transparent Rectangle	Draws a transparent rectangle
Transparent Circle	Draws a transparent circle
Eraser	Erases the objects on the designer area
Polygon	Draws a polygon
Gray Circle	Draws a gray circle
Light Gray Circle	Draws a light gray circle
Dark Gray Circle	Draws a dark gray circle

Gray Rectangle	Draws a gray rectangle
Light Gray Rectangle	Draws a light gray rectangle
Dark Gray Rectangle	Draws a dark gray rectangle
Copy Tree Items	Copies the tree items on the designer area
Text	Draws a text string

5. Color

Edit color	Presents a color dialog box for selecting a color

6. Windows

New Window	Opens the new window
Tile	Tiles the windows
Cascade	Cascades the windows
Arrange Icons	Arranges the icons

7. Font

Edit Font	Presents a font dialog box for selecting a font

8. Help

About Form Builder	Presents the version information and the copyright information

DOCKING WINDOWS

Designer Window	User designs form layouts in the designer window
Tree Window	Displays and maintains designer tree and grid tree
Grid Window	Displays and maintains the grid control

TAB CONTROLS

Designer Tab	Activates the designer tree view in Tree docking window
Grid Tab	Activates the grid tree view in Tree docking window

3.3.2.2 Hardware Interfaces

There are no hardware interfaces involved in this system.

3.3.2.3 Software Interfaces

The system is designed on Win32 platform so it supports Microsoft Windows 98,Me,2000 and XP (Whistler). However its graphics rendering capabilities are viewed best in Microsoft XP interface.

3.3.3 Non Functional Requirements

- The system should be flexible enough to change with the change in customer desires.
- The system should not crash at any time due to unavailability of reaction to any event.

3.3.4 Limitations and Constraints

Time

The time allocated for the completion of project was 4 months.

Knowledge

We had a very little knowledge of XML and its related technologies. Moreover we were not proficient in Visual C++ 6.0.

Information

Information seeking is really a hard task. We had to consult some very experience programmers in this regard. They really provided us valuable information about the design of Form Builders.

Client Oriented Constraints

Every client wants to have certain benefits from a new system that act like a constraint in developing a software system.

- Cost Reduction.
- Better Use Of Resources.
- Time Saving.
- Efficient Product.

User Oriented Constraints

Making it menu driven, making it comparatively same job as they are doing right now.

Design Constraints

- Fulfilling All Functions Desired.
- Making System Smooth.
- Bitmaps are not supported in SVG format so there must be a programming turn around to support them.

3.4. Future Extension

Following extensions are proposed in the Form Builder:

1. Animations can be incorporated in the form builder. A large number of hottest animation software ,like Flash 5.0 or 3d Studio Max, use vector graphics technology for the creation and management of animations. SVG supports the use of animation because it is a vector graphics based technology. Since Form Builder uses SVG for storing and retrieving the data so it is very easy to incorporate animations in the online forms.
2. Form Builder can be further extended to support various other XML based presentation technologies. For example, support for the XSLT can be incorporated or Cascading Style sheets can be supported by Form Builder.
3. Form Builder can be extended to support some very famous graphics format like jpeg, gif, targa, cdr , psd. Presently, Form Builder only supports bitmaps.
4. Form Builder can be used to support Adobe's famous Portable Document Format commonly known as PDF. A large amount of data on the web is available in the form of PDF supported pages so if Form Builder supports PDF then its interoperability and usefulness will be increased to a greater extent.
5. Form Builder can be developed as a Application Builder for the famous print streams like IBM's AFP or Xerox's VIPP. Incorporating the support for these famous print stream formats will certainly make the Form Builder a valuable variable printing tool.

Section 4

CONCEPT OF XML

4.1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

4.2. Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.

3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

4.3. Structuring Data

Structured data includes things like spreadsheets, address books, configuration parameters, financial transactions, and technical drawings. XML is a set of rules for designing text formats that let you structure your data. XML is not a programming language. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids common pitfalls in language design: it is extensible, platform-independent, and it supports internationalization and localization. XML is fully Unicode-compliant.

4.4. XML and HTML

Like HTML, XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form `name="value"`). While HTML specifies what each tag and attribute means, and often how the text between them will look in a browser, XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, "<p>" in an XML file, do not assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person, a p... (and who says it has to be a word with a "p"?).

4.5. XML as text

Programs that produce spreadsheets, address books, and other structured data often store that data on disk, using either a binary or text format. One advantage of a text format is that it allows people, if necessary, to look at the data without the program that produced it. Text formats also allow developers to more easily debug applications. Like HTML, XML files are text files that people shouldn't have to read, but may when the need arises. Less like HTML, the rules for XML files are strict. A forgotten tag, or an attribute without quotes makes an XML file unusable, while in HTML such practice is tolerated and is often explicitly allowed. The official XML specification forbids applications from trying to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and report an error.

4.6. Design of XML

Since XML is a text format and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the designers of XML. The advantages of a text format are evident and the disadvantages can usually be compensated at a different level. Disk space is less expensive than it used to be, and compression programs like zip and gzip can compress files very well and very fast. In addition, communication protocols such as modem protocols and HTTP/1.1, the core protocol of the Web, can compress data on the fly, saving bandwidth as effectively as a binary format.

4.7. Family of technologies

XML 1.0 is the specification that defines what "tags" and "attributes" are. Beyond XML 1.0, "the XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. *Xlink* describes a standard way to add hyperlinks to an XML file. *XPointer* and *XFragments* are syntaxes in development for pointing to parts of an XML document. An XPointer is a bit like a URL,

but instead of pointing to documents on the Web, it points to pieces of data inside an XML file. *CSS*, the style sheet language, is applicable to XML as it is to HTML. *XSL* is the advanced language for expressing style sheets. It is based on *XSLT*, a transformation language used for rearranging, adding and deleting tags and attributes. The *DOM* is a standard set of function calls for manipulating XML (and HTML) files from a programming language. *XML Schemas 1* and *2* help developers to precisely define the structures of their own XML-based formats. There are several more modules and tools available or under development.

4.8. Development of XML

Development of XML started in 1996 and has been a W3C Recommendation since February 1998, which may make anyone suspect that this is rather immature technology. In fact, the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. The development of HTML started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, and vastly more regular and simple to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

4.9. USE OF XHTML

There is an important XML application that is a document format: W3C's XHTML, the successor to HTML. XHTML has many of the same elements as HTML. The syntax has been changed slightly to conform to the rules of XML. A document that is "XML-based" inherits the syntax from XML and restricts it in certain ways (e.g, XHTML allows "<p>", but not "<r>"); it also adds meaning to that syntax (XHTML says that "<p>" stands for "paragraph", and not for "price", "person", or anything else).

4.10. Modular Approach

XML allows to define a new document format by combining and reusing other formats. Since two formats developed independently may have elements or attributes with the same name, care must be taken when combining those formats (does "<p>" mean "paragraph" from this format or "person" from that one?). To eliminate name confusion when combining formats, XML provides a *namespace* mechanism. XSL and RDF are good examples of XML-based formats that use namespaces. *XML Schema* is designed to mirror this support for modularity at the level of defining XML document structures, by making it easy to combine two schemas to produce a third which covers a merged document structure.

4.11. Basis for RDF and the Semantic Web

W3C's Resource Description Framework (RDF) is an XML text format that supports resource description and metadata applications, such as music playlists, photo collections, and bibliographies. For example, RDF might let you identify people in a Web photo album using information from a personal contact list; then your mail client could automatically start a message to those people stating that their photos are on the Web. Just as HTML integrated documents, menu systems, and forms applications to launch the original Web, RDF integrates applications and agents into one Semantic Web. Just like people need to have agreement on the meanings of the words they employ in their communication, computers need mechanisms for agreeing on the meanings of terms in order to communicate effectively. Formal descriptions of terms in a certain area (shopping or manufacturing, for example) are called ontologies and are a necessary part of the Semantic Web. RDF, ontologies, and the representation of meaning so that computers can help people do work are all topics of the Semantic Web Activity.

4.12. Platform-independence and well-supported

By choosing XML as the basis for a project, you gain access to a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs and procedures that manipulate it, and there are many tools available and many people who can help you. And since XML is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. *XML isn't always the best solution, but it is always worth considering.*

Section 5

CONCEPT OF SVG

5.1. Introduction

About SVG

SVG is a language for describing two-dimensional graphics in XML [XML10]. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting.

Sophisticated applications of SVG are possible by use of a supplemental scripting language which accesses SVG Document Object Model (DOM), which provides complete access to all elements, attributes and properties. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on XHTML and SVG elements simultaneously within the same Web page.

SVG is a language for rich graphical content. For accessibility reasons, if there is an original source document containing higher-level structure and semantics, it is recommended that the higher-level information be made available somehow, either by making the original source document available, or making an alternative version available in an alternative format which conveys the higher-level information, or by using SVG's facilities to include the higher-level information within the SVG content.

5.1.2 SVG MIME type, file name extension and Macintosh file type

The MIME type for SVG is "image/svg+xml" . The W3C will register this MIME type around the time when SVG is approved as a W3C Recommendation.

It is recommended that SVG files have the extension ".svg" (all lowercase) on all platforms. It is recommended that gzip-compressed SVG files have the extension ".svgz" (all lowercase) on all platforms.

It is recommended that SVG files stored on Macintosh HFS file systems be given a file type of "svg " (all lowercase, with a space character as the fourth letter). It is recommended that gzip-compressed SVG files stored on Macintosh HFS file systems be given a file type of "svgz" (all lowercase).

5.1.3 SVG Namespace, Public Identifier and System Identifier

The following are the SVG 1.0 namespace, public identifier and system identifier:

SVG Namespace:

http://www.w3.org/2000/svg

Public Identifier for SVG 1.0:

PUBLIC "-//W3C//DTD SVG 1.0//EN"

System Identifier for SVG 1.0:

http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd

The following is an example document type declaration for an SVG document:

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"

"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

5.1.4 Compatibility with Other Standards Efforts

SVG leverages and integrates with other W3C specifications and standards efforts. By leveraging and conforming to other standards, SVG becomes more powerful and makes it easier for users to learn how to incorporate SVG into their Web sites.

The following describes some of the ways in which SVG maintains compatibility with, leverages and integrates with other W3C efforts:

- SVG is an application of XML and is compatible with the "Extensible Markup Language (XML) 1.0".
- SVG is compatible with the "Namespaces in XML”
- SVG utilizes "XML Linking Language (XLink)" [XLINK] for URI referencing and requires support for base URI specifications defined in "XML Base"
- SVG's syntax for referencing element IDs is a compatible subset of the ID referencing syntax in "XML Pointer Language (XPointer)"
- SVG content can be styled by either CSS or XSL.
- SVG supports relevant properties and approaches common to CSS and XSL, plus selected semantics and features of CSS.
- External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0”.
- SVG includes a complete Document Object Model (DOM) and conforms to the "Document Object Model (DOM) level 1".
- The SVG DOM has a high level of compatibility and consistency with the HTML DOM . Additionally, the SVG DOM supports [DOM2], including the CSS object model and event handling.

- SVG incorporates some features and approaches that are part of the "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", including the 'switch' element and the system Language attribute.
- SVG's animation features were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification . SVG's animation features incorporate and extend the general-purpose XML animation capabilities described in the "SMIL Animation" specification .
- SVG has been designed to allow future versions of SMIL to use animated or static SVG content as media components.
- SVG attempts to achieve maximum compatibility with both HTML 4 [HTML4] and XHTML(tm) 1.0 [XHTML]. Many of SVG's facilities are modeled directly after HTML, including its use of CSS [CSS2], its approach to event handling, and its approach to its Document Object Model [DOM2].
- SVG is compatible with W3C work on internationalization.
- SVG is compatible with W3C work on Web Accessibility [WAI].

In environments which support [DOM2] for other XML grammars (e.g., XHTML [XHTML]) and which also support SVG and the SVG DOM, a single scripting approach can be used simultaneously for both XML documents and SVG graphics, in which case interactive and dynamic effects will be possible on multiple XML namespaces using the same set of scripts.

5.2. Concepts

5.2.1 Explaining the name: SVG

SVG stands for Scalable Vector Graphics, an XML grammar for stylable graphics, usable as an XML namespace.

Scalable

To be scalable means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size. On the Web, scalable means that a particular technology can grow to a large number of files, a large number of users, a wide variety of applications. SVG, being a graphics technology for the Web, is scalable in both senses of the word.

SVG graphics are scalable to different display resolutions, so that for example printed output uses the full resolution of the printer and can be displayed at the same size on screens of different resolutions. The same SVG graphic can be placed at different sizes on the same Web page, and re-used at different sizes on different pages. SVG graphics can be magnified to see fine detail, or to aid those with low vision.

SVG graphics are scalable because the same SVG content can be a stand-alone graphic or can be referenced or included inside other SVG graphics, thereby allowing a complex illustration to be built up in parts, perhaps by several people. The symbol, marker and font capabilities promote re-use of graphical components, maximize the advantages of HTTP caching and avoid the need for a centralized registry of approved symbols.

Vector

Vector graphics contain geometric objects such as lines and curves. This gives greater flexibility compared to raster-only formats (such as PNG and JPEG) which have to store information for every pixel of the graphic. Typically, vector formats can also integrate raster images and can combine them with vector information such as clipping paths to produce a complete illustration; SVG is no exception.

Since all modern displays are raster-oriented, the difference between raster-only and vector graphics comes down to where they are rasterized; client side in the case of vector graphics, as opposed to already rasterized on the server. SVG gives control over

the rasterization process, for example to allow anti-aliased artwork without the ugly aliasing typical of low quality vector implementations. SVG also provides client-side raster filter effects, so that moving to a vector format does not mean the loss of popular effects such as soft drop shadows.

Graphics

Most existing XML grammars represent either textual information, or represent raw data such as financial information. They typically provide only rudimentary graphical capabilities, often less capable than the HTML 'img' element. SVG fills a gap in the market by providing a rich, structured description of vector and mixed vector/raster graphics; it can be used stand-alone, or as an XML namespace with other grammars.

XML

XML, a W3C Recommendation for structured information exchange, has become extremely popular and is both widely and reliably implemented. By being written in XML, SVG builds on this strong foundation and gains many advantages such as a sound basis for internationalization, powerful structuring capability, an object model, and so on. By building on existing, cleanly-implemented specifications, XML-based grammars are open to implementation without a huge reverse engineering effort.

Namespace

It is certainly useful to have a stand-alone, SVG-only viewer. But SVG is also intended to be used as one component in a multi-namespace XML application. This multiplies the power of each of the namespaces used, to allow innovative new content to be created. For example, SVG graphics may be included in a document which uses any text-oriented XML namespace - including XHTML. A scientific document, for example, might also use MathML for mathematics in the document. The combination of SVG and SMIL leads to interesting, time based, graphically rich presentations.

SVG is a good, general-purpose component for any multi-namespace grammar that needs to use graphics.

Stylable

The advantages of style sheets in terms of presentational control, flexibility, faster download and improved maintenance are now generally accepted, certainly for use with text. SVG extends this control to the realm of graphics.

The combination of scripting, DOM and CSS is often termed "Dynamic HTML" and is widely used for animation, interactivity and presentational effects. SVG allows the same script-based manipulation of the document tree and the style sheet.

2.2 Important SVG concepts

Graphical Objects

With any XML grammar, consideration has to be given to what exactly is being modeled. For textual formats, modeling is typically at the level of paragraphs and phrases, rather than individual nouns, adverbs, or phonemes. Similarly, SVG models graphics at the level of graphical objects rather than individual points.

SVG provides a general path element, which can be used to create a huge variety of graphical objects, and also provides common basic shapes such as rectangles and ellipses. These are convenient for hand coding and may be used in the same ways as the more general path element. SVG provides fine control over the coordinate system in which graphical objects are defined and the transformations that will be applied during rendering.

Symbols

It would have been possible to define some standard symbols that SVG would provide. There would always be additional symbols for electronics, cartography, flowcharts, etc., that people would need that were not provided until the "next version". SVG allows users to create, re-use and share their own symbols without requiring a centralized registry. Communities of users can create and refine the symbols that they need, without having to ask a committee. Designers can be sure exactly of the graphical appearance of the symbols they use and not have to worry about unsupported symbols.

Symbols may be used at different sizes and orientations, and can be restyled to fit in with the rest of the graphical composition.

Raster Effects

Many existing Web graphics use the filtering operations found in paint packages to create blurs, shadows, lighting effects and so on. With the client-side rasterization used with vector formats, such effects might be thought impossible. SVG allows the declarative specification of filters, either singly or in combination, which can be applied on the client side when the SVG is rendered. These are specified in such a way that the graphics are still scalable and displayable at different resolutions.

Fonts

Graphically rich material is often highly dependent on the particular font used and the exact spacing of the glyphs. In many cases, designers convert text to outlines to avoid any font substitution problems. This means that the original text is not present and thus searchability and accessibility suffer. In response to feedback from designers, SVG includes font elements so that both text and graphical appearance are preserved.

Animation

Animation can be produced via script-based manipulation of the document, but scripts are difficult to edit and interchange between authoring tools is harder. Again in response to feedback from the design community, SVG includes declarative animation elements which were designed collaboratively by the SVG and SYMM Working Groups. This allows the animated effects common in existing Web graphics to be expressed in SVG.

5.2.3 Options for using SVG in Web pages

There are a variety of ways in which SVG content can be included within a Web page. Here are some of the options:

- A stand-alone SVG Web page
  In this case, an SVG document (i.e., a Web resource whose MIME type is "image/svg+xml") is loaded directly into a user agent such as a Web browser. The SVG document is the Web page that is presented to the user.
- Embedding by reference
  In this case, a parent Web page references a separately stored SVG document and specifies that the given SVG document should be embedded as a component of the parent Web page. For HTML or XHTML, here are three options:
    - The HTML/XHTML 'img' element is the most common method for using graphics in HTML pages. For faster display, the width and height of the image can be given as attributes. One attribute that is required is alt, used to give an alternate textual string for people browsing with images off, or who cannot see the images. The string cannot contain any markup. A longdesc attribute lets you point to a longer description - often in HTML - which can have markup and richer formatting.
    - The HTML/XHTML 'object' element can contain other elements nested within it, unlike 'img', which is empty. This means that several different

formats can be offered, using nested 'object' elements, with a final textual alternative (including markup, links, etc). The outermost element which can be displayed will be used.

  - The HTML/XHTML 'applet' element which can invoke a Java applet to view SVG content within the given Web page. These applets can do many things, but a common task is to use them to display images, particularly ones in unusual formats or which need to be presented under the control of a program for some other reason.

- Embedding inline
  In this case, SVG content is embedded inline directly within the parent Web page. An example is an XHTML Web page with an SVG document fragment textually included within the XHTML.
- External link, using the HTML 'a' element
  This allows any stand-alone SVG viewer to be used, which can (but need not) be a different program to that used to display HTML. This option typically is used for unusual image formats.
- Referenced from a CSS2 or XSL property
  When a user agent supports CSS-styled XML content or XSL Formatting Objects and the user agent is a Conforming SVG Viewer, then that user agent must support the ability to reference SVG resources wherever CSS or XSL properties allow for the referencing of raster images, including the ability to tile SVG graphics wherever necessary and the ability to composite the SVG into the background if it has transparent portions. Examples include the 'background-image' and 'list-style-image' properties that are included in both CSS and XSL.

Section 6

CONCEPT OF FLAT FILE

6.1 Introduction

Flat files are simply text files saved with an extension of *.flt. These files are the simplest and oldest forms of storing databases. In Form Builders, flat files are used that hold all the data. As Form Builders are made compatible for printing variable data, this compatibility can be achieved simply by using flat files. To provide printing solutions, a technology of XML named as PPML (Personalized Printing Markup Language) is used. PPML is an XML technology that uses tags in the same way as they are used in SVG. PPML, to provide printing solutions, use flat files as databases. When designs are saved in the memory of the printer then the data that has to be printed is send to the printer in the form of flat files and this solution is provided by using PPML as Form Builder had to be made compatible with printing variable data it had to use the same sort of database as used by PPML. PPML can easily be incorporated in Form Builder as Form Builder use the same sort of database as used by PPML, which is in flat file format.

6.2 Use of Flat File in Form Builder

Flat Files are of various types. Flat Files that Form Builder uses are of the simplest type that contains simple data in the form of text without titles. The reason for using Flat Files with out titles is that flat file has to be sent to the printer and the data written in flat file is printed according to the designed form saved into the memory of the printer. If flat files with titles were used, then it would cause an error in printing as printer would consider the titles as data and would print those titles too. In Form Builders, simple flat files were used and the titles were given by simply opening data in flat file into a grid, later when PPML has to be used with Form Builder the variables will be declared for each title given to the field of flat file data in Form Builder and that variable will be attached to the place where that data will be printed for that title and this variable declaration will be done using Visual C++ and PPML. The printing will be done with the help of those variables and data will recognize its place in the form with the help of the variable attached with it.

6.3. Structure of Flat File

Broadly speaking, Flat Files can be divided into two categories, fixed and variable size. The difference between the two categories of flat files is the lengths of the fields. Lengths of the fields are fixed in case of fixed sized flat files and variable in the case of variable sized flat files. All the flat files contain data in the form of documents, each document is separated by “*” or “**” or “***” or any other delimiter can be used as declared by the user. Each document then contains records having unique record id separated by newline character. A document may contain several records having same id. Each record then contains different fields. Data lies in those fields. In case of fixed size flat files, fields are separated by their lengths while in case of variable size, they are separated by commas. Each field of records having same id are of same type, similarly types of each record and fields in all the documents are entirely identical. If record id 1 of first document had three fields, then all the records having id 1 in the first documents will have three fields. Similarly, in other documents, record id 1 will have three fields and types of those fields in all documents will also be the same.

Example:

An Example of flat file having different documents and records is given below

1,S.M. Khaliq-ur-RahmanRaazi,Male,None,50279689207,6110118015325

2,Syed Muhammad Khalil-ur-Rahman,X517R3

3,03/09/1979,08/08/2001,31/07/2010

4,H#450 St.#40 I-8/2 Islamabad,A-24/N North Nazimabad Karachi

*

1,Waseem Imran,Male,None,91827364548,1928374656473

2,Mohammad Akram,Y110T3

3,16/10/1979,10/08/2001,,16/09/2010

4,H#42 Tipu Rd. Rawalpindi,H#42 Tipu Rd. Rawalpindi

*

1,Junaid Arshed,Male,None,19283746564,0918273645738

2,Arshed Parvez,Z225M1

3,26/11/1980,10/09/2001,25/12/2009

4,B-352/D Satellite Town Rawalpindi,B-352/D Satellite Town Rawalpindi

*

Section 7

Object Oriented Analysis

USE CASE ANALYSIS

Two use cases are involved in the form builder. These are

1. Design Form Layout.
2. Flat File Opener & Field Title Assignment.

1. Design Form Layout

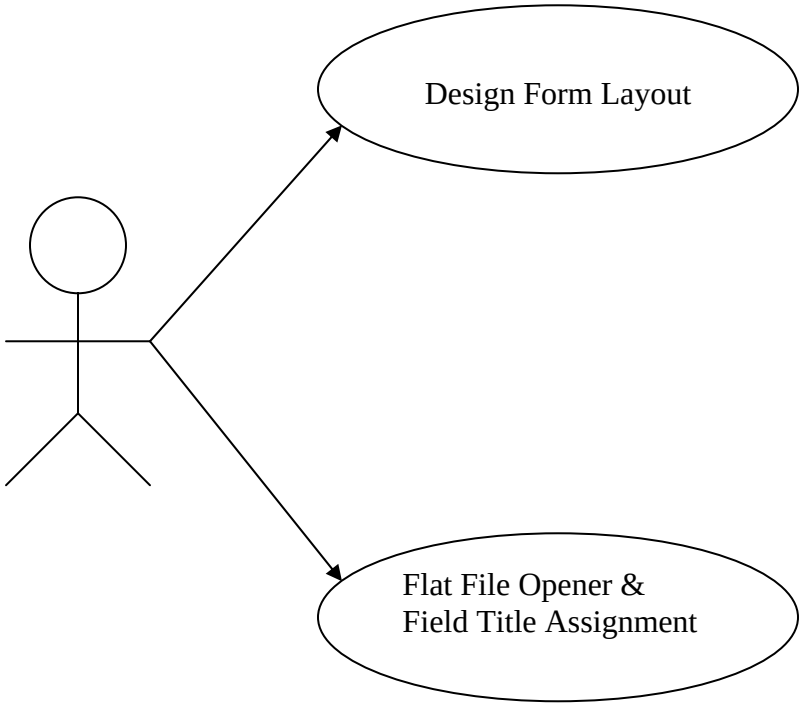
In this use case, a user designs a form layout with the help of various drawing tools. To facilitate the designing, the environment of the Form Builder is very user friendly. Form Builder maintains a tree known as “Designer Tree”. There is an empty area in form builder on which user can draw different kinds of shapes and import bitmaps. The objects drawn on designer area are maintained in the nodes of designer tree. The designer tree offers the users a very easy way to navigate through various drawn objects. The drawn objects constitute the form layout. The designing of the form layout is an essential procedure in form building. Variable data from a flat file is mapped onto the form layout. The form is ready for displaying or printing after the mapping of variable data onto the form layout.

2. Flat File Opener & Field Title Assignment

In this use case, a user opens a flat file in a grid. Flat file is a simple text file. It is a very simple kind of database with virtually no metadata. Data from a large database residing on a main frame is acquired once and downloaded in a simple text file known as a flat file. The data is not stored in a very systematic way. It is not possible for a user to comprehend the data. The fields are separated by a predefined delimiter and records are differentiated by unique ids. These fields and records constitute a document. Each document is then separated by a delimiter. A flat file can consist of a very large number of documents and the size of a flat file is usually in Giga bytes. The flat file is the source of variable data which is to be printed on the form layout. Form Builder opens the flat file in a grid.

The user assigns titles to the fields of the flat file. These titles are maintained by a tree view known as “Grid View”. The user can navigate through the titles assigned to the fields in the grid with the help of the Grid View. These titles are then dragged and dropped onto the various segments of the form layout. Now the form is ready for displaying or printing.

USE CASE DIAGRAM

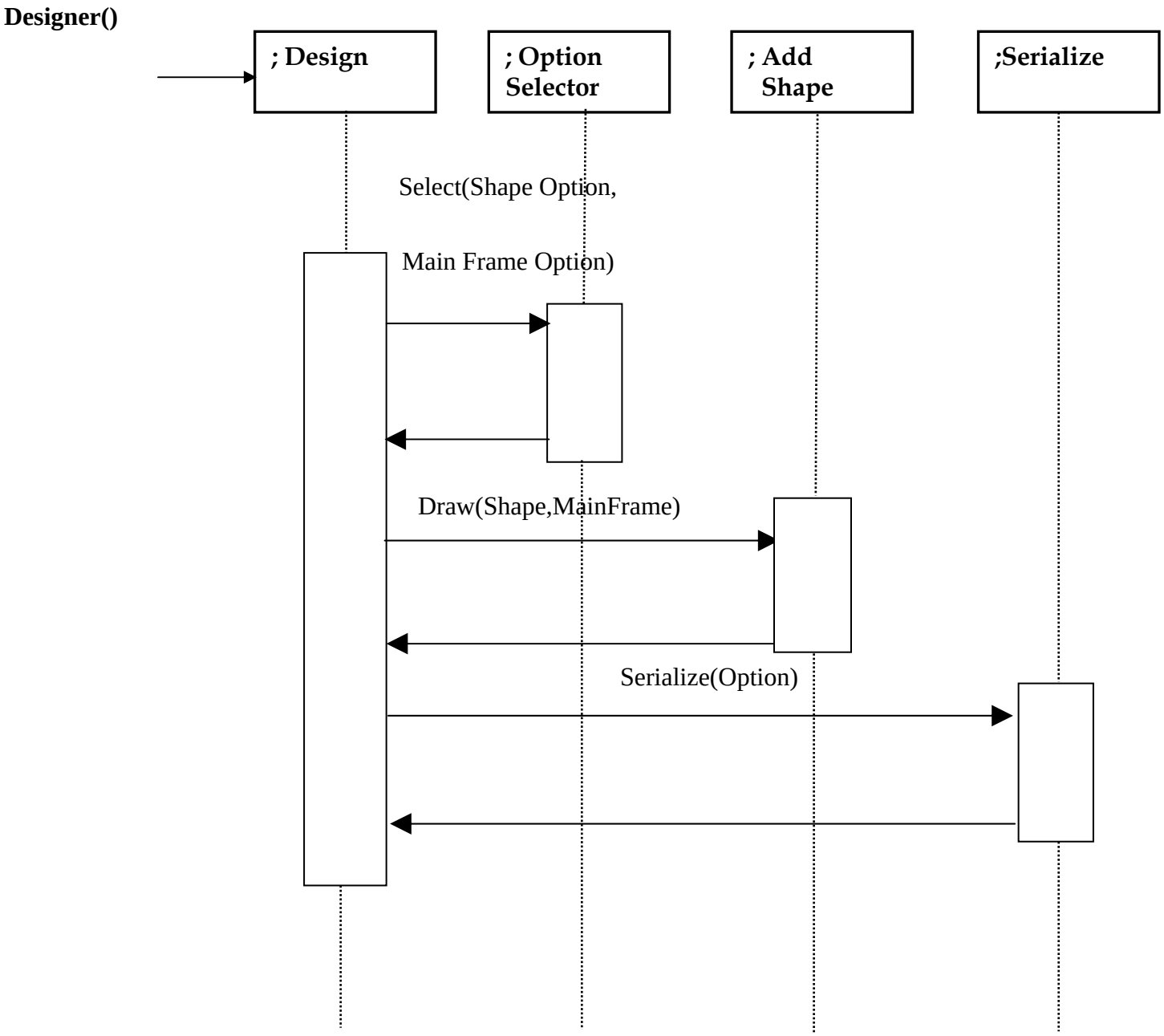


Section 8

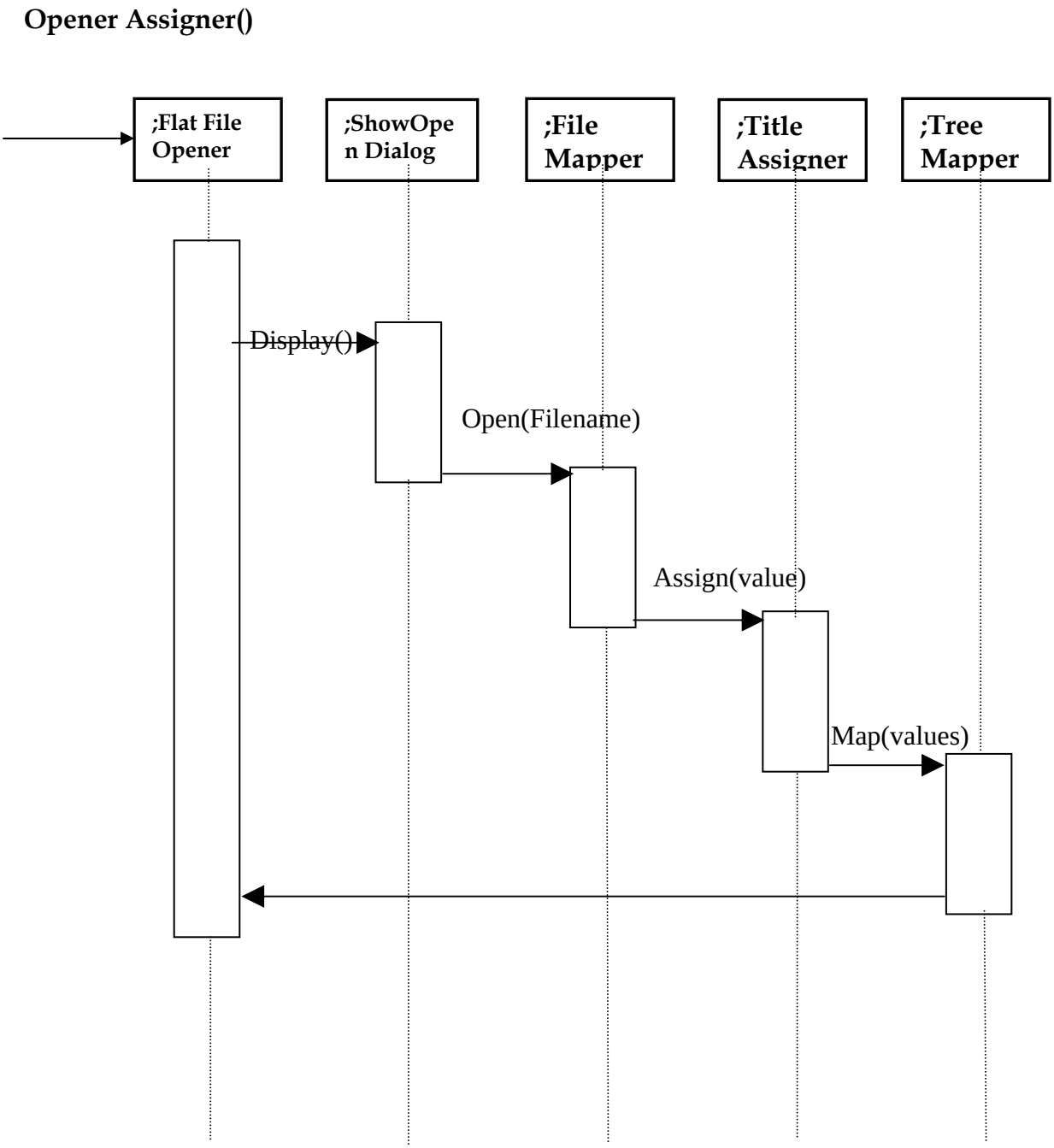
Object Oriented Analysis

SEQUENCE DIAGRAMS

Design Form Layout



Flat File Opener & Field Title Assignment



Section 9

Object Oriented Analysis

PROBLEM DOMAIN

OBJECT LIST

Candidate List

Designer Tree
Docking Window
Form Builder App
Form Builder Doc
Form Builder View
Form Tree
Grid Window
Grid View
Main Frame
Open Dialog
Line
Rectangle
Circle
Polygon
Pencil
Text
Sizing Tab Control Bar
Shapes
Grid
XML Opener
XML Serializer

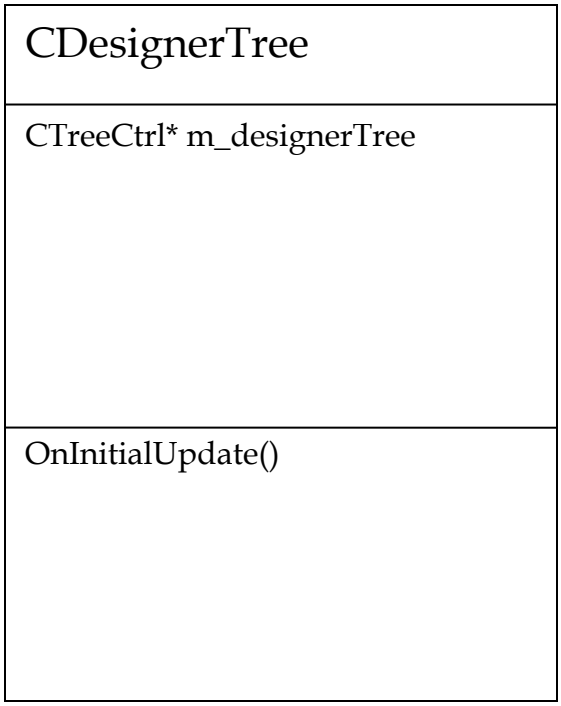
Selected List

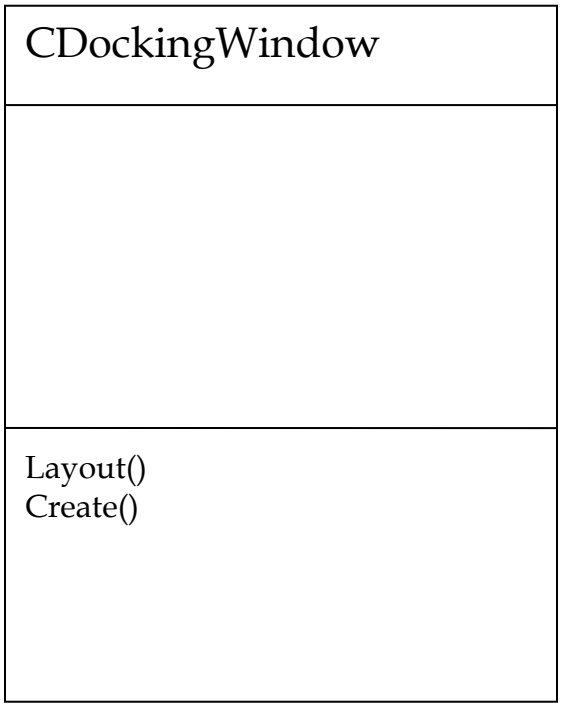
Form Builder App
Form Builder Doc
Form Builder View
Form Tree
Grid Window
Grid View
Main Frame
Open Dialog
Sizing Tab Control Bar
Shapes
Grid

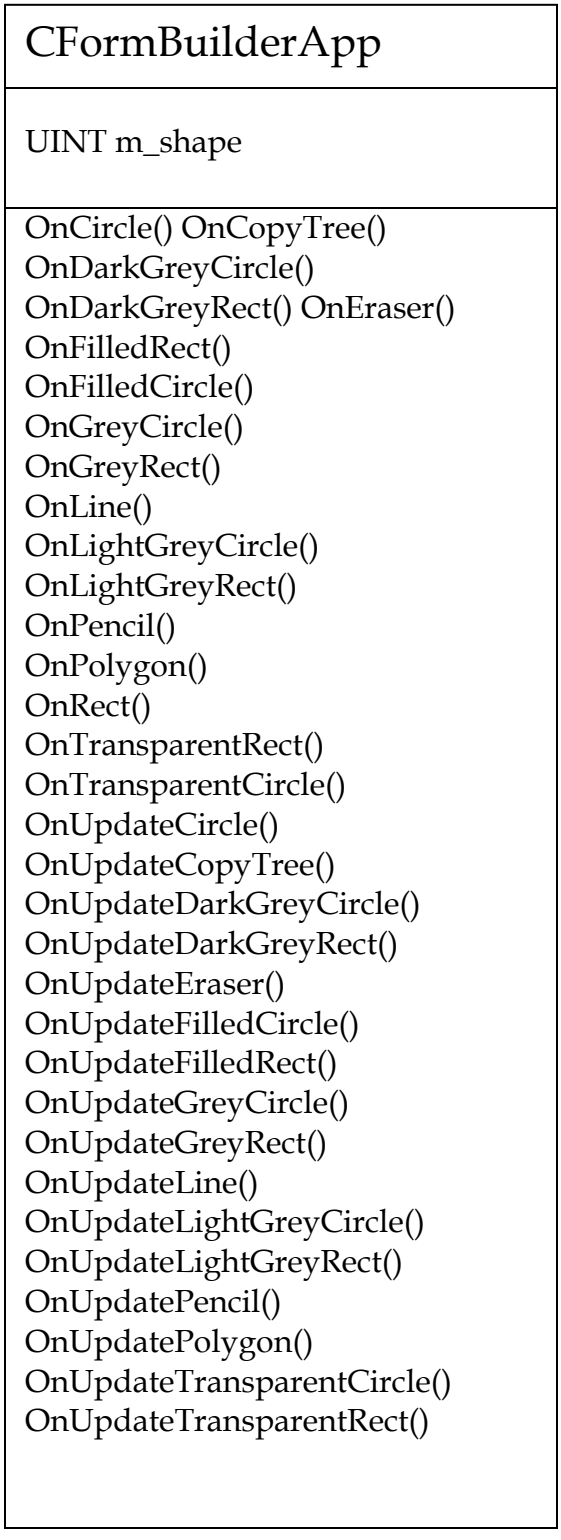
Section 10

Object Oriented Design

Class List

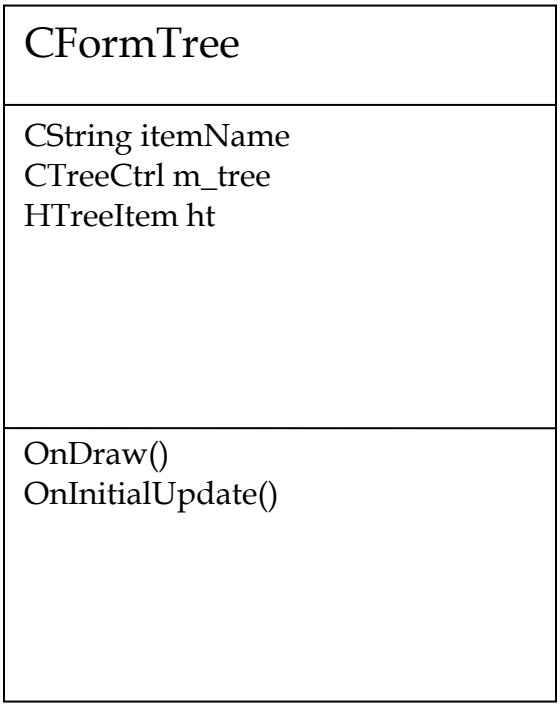

CDesignerTree
CTreeCtrl* m_designerTree
OnInitialUpdate()

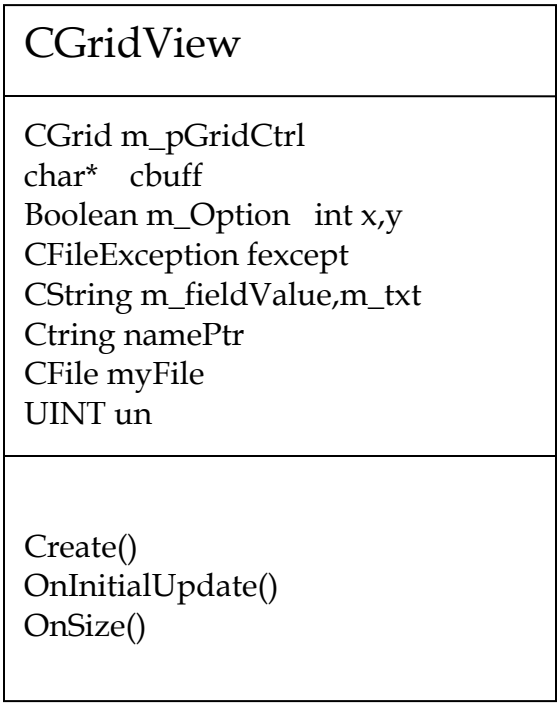

CDockingWindow
Layout()
Create()

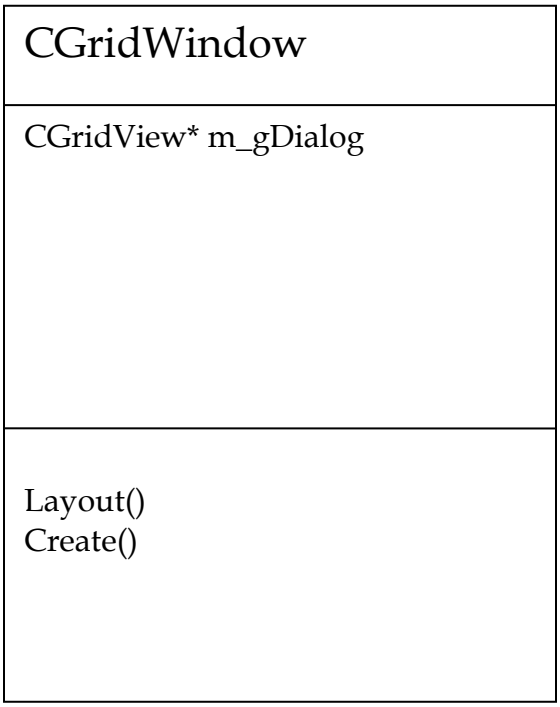

CFormBuilderApp
UINT m_shape
OnCircle() OnCopyTree()
OnDarkGreyCircle()
OnDarkGreyRect() OnEraser()
OnFilledRect()
OnFilledCircle()
OnGreyCircle()
OnGreyRect()
OnLine()
OnLightGreyCircle()
OnLightGreyRect()
OnPencil()
OnPolygon()
OnRect()
OnTransparentRect()
OnTransparentCircle()
OnUpdateCircle()
OnUpdateCopyTree()
OnUpdateDarkGreyCircle()
OnUpdateDarkGreyRect()
OnUpdateEraser()
OnUpdateFilledCircle()
OnUpdateFilledRect()
OnUpdateGreyCircle()
OnUpdateGreyRect()
OnUpdateLine()
OnUpdateLightGreyCircle()
OnUpdateLightGreyRect()
OnUpdatePencil()
OnUpdatePolygon()
OnUpdateTransparentCircle()
OnUpdateTransparentRect()

CFormBuilderDoc
CObArray m_oaShapes ColorRef m_color int stockObject
AddShape() DeleteContents() GetNumShape() OnColorPallette() OnEditClearAll() OnEditUndo() OnFileDialog() OnUpdateClearAll() OnUpdateEdit() Serialize()

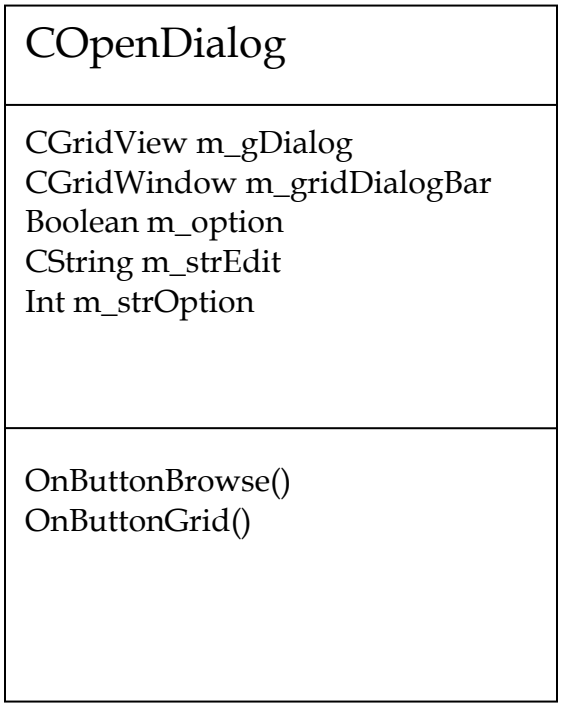
CFormBuilderView
CPoint newPoint CPoint oldPoint
OnArrow() OnDraw()

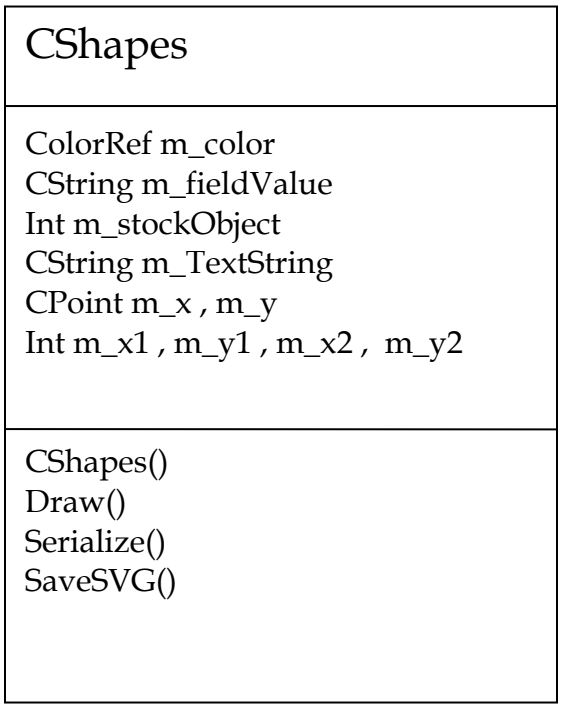

CFormTree
CString itemName
CTreeCtrl m_tree
HTreeItem ht
OnDraw()
OnInitialUpdate()

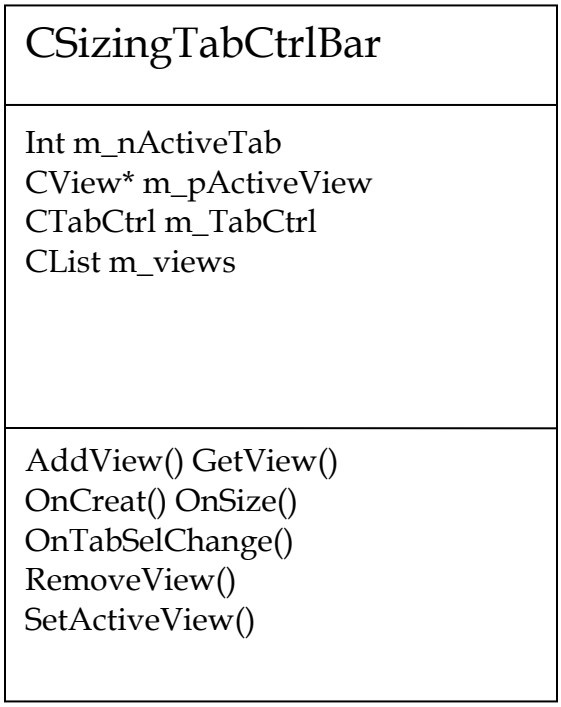

CGridView
CGrid m_pGridCtrl
char* cbuff
Boolean m_Option int x,y
CFileException fexcept
CString m_fieldValue,m_txt
Ctring namePtr
CFile myFile
UINT un
Create()
OnInitialUpdate()
OnSize()

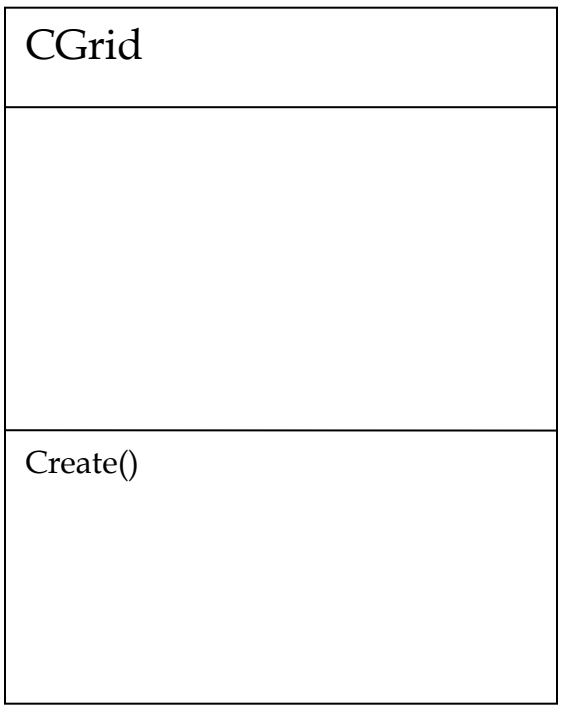

CGridWindow
CGridView* m_gDialog
Layout()
Create()

CMainFrame
char* cbuff char ch char* characterstoWrite DWORD dwStyle CFileException fexcept int iCount Boolean m_boolTree int m_bufferSize CFormTree m_cDialog Boolean m_childVisible int m_colCount CGridView m_gDialog CGridWindow m_gridDialogBar Boolean m_gridPressure Boolean m_isTreePresent COpenDialog m_MyDialog CString m_NamePtr Int m_noofRoots CGridCtrl* m_pGridCtrl Int m_rowCount UINT m_Shape CToolBar m_shapeTooldBar CString m_strEdit , m_txt CTreeCtrl m_tree CDockingWindow m_wndDialogBar CStatusBar m_wndStatusBar CSizingTabCtrlBar m_stcBar CToolBar m_wndToolBar CFile myFile CString namePtr CString** value
mapFile() mapTree() OnCreate() OnDoc() OnMapTree() OnOpenGrid() OnSaveGrid() OnOpenSVG() OnSaveSVG() OnUpdateViewShapeToolBar() OnViewShapeToolBar()


COpenDialog
CGridView m_gDialog
CGridWindow m_gridDialogBar
Boolean m_option
CString m_strEdit
Int m_strOption
OnButtonBrowse()
OnButtonGrid()


CShapes
ColorRef m_color
CString m_fieldValue
Int m_stockObject
CString m_TextString
CPoint m_x , m_y
Int m_x1 , m_y1 , m_x2 , m_y2
CShapes()
Draw()
Serialize()
SaveSVG()


CSizingTabCtrlBar
Int m_nActiveTab
CView* m_pActiveView
CTabCtrl m_TabCtrl
CList m_views
AddView() GetView()
OnCreat() OnSize()
OnTabSelChange()
RemoveView()
SetActiveView()


CGrid
Create()

Section 11

Object Oriented Design

CLASS RESPONSIBILITY

COLLABORATION CARDS

Class name : CDesignerTree	
Base : CTreeView	
Derived: none	
Responsibilities	Collaborators
1.Maps the drawn shapes on designer tree. **2.Changes the attributes of the drawn shapes.**	**CFormBuilderDoc**

DESCRIPTION

CDesignerTree is derived form CTreeView which is a Microsoft Foundation Class. CDesignerTree has a very important responsibility of mapping the drawn shapes, which are drawn on designer area, on the designer tree.During this operation, it collaborates with the CFormBuilderDoc class.Another important responsibility of this class is to change the attributes of the drawn shapes. We can change the size, length, width and color of the shapes in designer area and this class actually perform the operation on the backend.

Class name : CDockingWndow	
Base : CCtrlBar	
Derived: CSizingTabCtrlBar	
Responsibilities	Collaborators
1.Creates the docking window **2.Changes the Layout of Docking window**	

DESCRIPTION

CDockingWindow is derived from CctrlBar which is a MFC class.CSizingTabCtrlBar is derived from this class. It creates the docking window in the designer area. This class also changes the layout of the docking window. It consists of two methods for performing the operations assigned to it.

Class name : CFormBuilderApp	
Base : CWinApp	
Derived: none	
Responsibilities	Collaborators
1.Maintains the message handlers of all shapes.	

DESCRIPTION

CFormBuilderApp is the application class which is formed by MFC for the management of the developing application. This class is derived from CWinApp which is a very important class for creating application on Win32 platform.None of the classes are further derived from CFormBuilderApp.It manages the messages which are originated from different events and shapes in the design area. It provides message handlers for responding to the originated messages. It has 33 message handler functions to respond to different events.

Class name : CFormBuilderDoc	
Base : CDocument	
Responsibilities	Collaborators
1.Add Shapes **2.Delete Contents** **3.GetShapes** **4.Maintains Colors** **5.Maintains edit and clearAll functions**	**CShape** **CFormBuilderView** **CFormBuilderApp**

DESCRIPTION

CFormBuilderDoc is the document class. It is derived from Cdocument which is a MFC class. CformBuilderDoc is actually the document portion of the Microsoft's Document/View architecture. It adds shapes to designer area. It deletes the contents. It also takes care of the colors and it maintains the operations of edit and clearAll functions.It has 8 functions to perform its required operations.

Class name : CFormBuilderView	
Base : CTreeView	
Derived: none	
Responsibilities	Collaborators
1. Maintains the view on mouse movements. **2. Maintains the form view on draw**	**CFormBuilderApp** **CFormBuilderDoc**

DESCRIPTION

CFormBuilderView is the view class. It is derived from CTreeView which is a MFC class. It is the view class which is the other important class in Document/View architecture. It maintains the view of the designer and application environment during the mouse movement.It perform this operation with the help of CFormBuilderApp. It also maintains the form view during the drawing. It performs this operation with the help of CFormBuilderDoc. It has two functions to perform its required operations.

Class name : CFormTree	
Base : CTreeView	
Derived: none	
Responsibilities	Collaborators
1. Maintains the form tree on initial update **2. Maintains the form tree on draw**	**CGridView** **CFormBuilderView**

DESCRIPTION

CFormTree is derived from CTreeView which is a MFC class.None of any other classes are derived from CFormTree.It maintains the states of form tree during the initial update with the help of CGridView class. It also maintains the state of the form tree during the drawing operation. It performs this operation with the help of CFormBuilderView. It has two functions to perform its operations.

Class name : CGridView	
Base : CView	
Derived: none	
Responsibilities	Collaborators
1.Creates the grid view **2.Maintains the grid view on initial update**	**CGrid** **CFormTree**

DESCRIPTION

CGridView is a very important class. It is derived from CView class which is a MFC class. None of any other classes are derived from CgridView. It has a one of the major responsibility to create the grid.It performs this operation with the help of CGrid class.It also maitains the grid view on initiail update with the help of CFormTree class. It performs the operations with the help of three functions.

Class name : CMainFrame	
Base : CMDIFrameWnd	
Derived: none	
Responsibilities	Collaborators
1. Maps the Flat file **2. Maps the data onto the tree** **3. Perform the tasks on opening the grid** **4. Performs the tasks on saving the grid** **5. Maintains the shape toolbar view** **6. Creates the docking window**	**CGridView** **CFormTree** **CGrid** **CDockingWindow**

DESCRIPTION

CMainFrame is derived from CMDIFrameWnd class which is a MFC class. None of any other classes are derived from CMainFrame class. CMainFrame has a major responsibility of mapping a flat file in the grid view. It performs this operation with the help of CGridView class. It also maps the data onto the tree with the help of CFormTree. It performs a very important operation of opening and saving the grid. It performs this operation with the of CGrid class. It also maintains the shape toolbar view.

Class name : COpenDialog	
Base : CDialog	
Derived: none	
Responsibilities	Collaborators
1. Opens the dialog box on clicking the browse button **2. Open the grid open dialog box on clicking the open grid button**	**CGridView**

DESCRIPTION

COpenDialog is derived from CDialog class which is a MFC class.None of any other classes are derived from COpenDialog.It opens the dialog box to browse through the list of flat files. It also opens the grid open dialog box on clicking on the open grid button. It performs this operation with the help of CGridView class. It has two functions to perform its operations.

Class name : CShapes	
Base : CObject	
Derived: none	
Responsibilities	Collaborators
1. Draws different shapes	**CFormBuilderApp, CFOrmBuilderDoc, CFormBuilderView**

DESCRIPTION

CShapes is a very important class. It is derived from CObject class which is a MFC class. None of any other classes are derived from CShapes class. It draws different shapes on the designer area. It performs this very important and pivotal task with the help of CFormBuilderApp, CFormBuilderDoc and CFormBuilderView class. It has two functions to give its desired output.

Class name : CSizingTabCtrlBar	
Base : CDockingWindow	
Derived: none	
Responsibilities	Collaborators
1. **Adds a view** 2. **Gets a view** 3. **Presents the corresponding view on clicking different tab selections**	**CFormTree,CDesignerTree**

DESCRIPTION

CSizingTabCtrlBar is derived from CDockingWindow class which is a generic class. None of any other class are derived from CSizingTabCtrlBar. It adds a view in the docking window. It performs this operation with the help of CformTree and CdesignerTree. It also gets the desired view and it switches between the corresponding views when a user clicks on different tab selections. It has 7 methods to perform its operations.

Class name : CGrid	
Base : CWnd	
Derived: none	
Responsibilities	Collaborators
1. Maintains the grid characteristics **2. Perform different kinds of functions associated with the grid**	**CMainFrame**

DESCRIPTION

CGrid is derived from CWnd class which is a MFC class. None of any other classes are derived from CGrid class. It has a very important task of maintaining the grid characteristics. It performs this task with the help of CMainFrame. It also performs different kinds of functions which are associated with the grid such as determining the number of rows and columns in the grid to accommodate the input data. It has one function to perform its task.

Class name : CGridWindow	
Base : CCtrlBar	
Derived: none	
Responsibilities	Collaborators
1.Creates the grid window **2.Forms the layout of the grid window** **3.Maintains the layout of the grid window**	**CGrid,CGridView**

DESCRIPTION

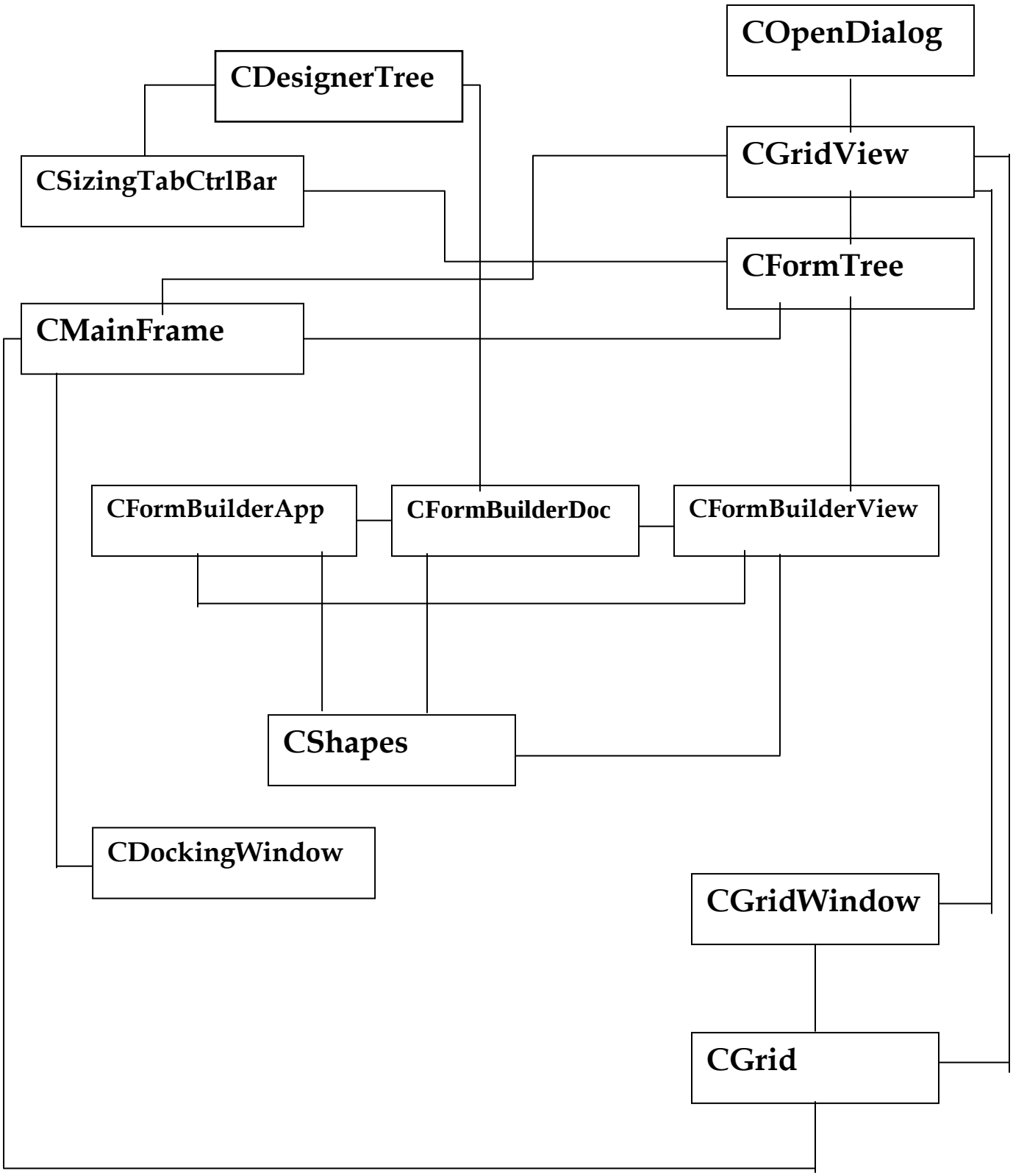
CGridWindow is derived from CCtrlBar which is a MFC class.None of any other classes are derived from this class. It creates the grid window with the help of CGrid and CGridView classes. It also forms and maitains the layout of grid window. It has two methods to perform its operations.

Section 12

Object Oriented Design

Class Relationship Diagram

Class Relationship Diagram

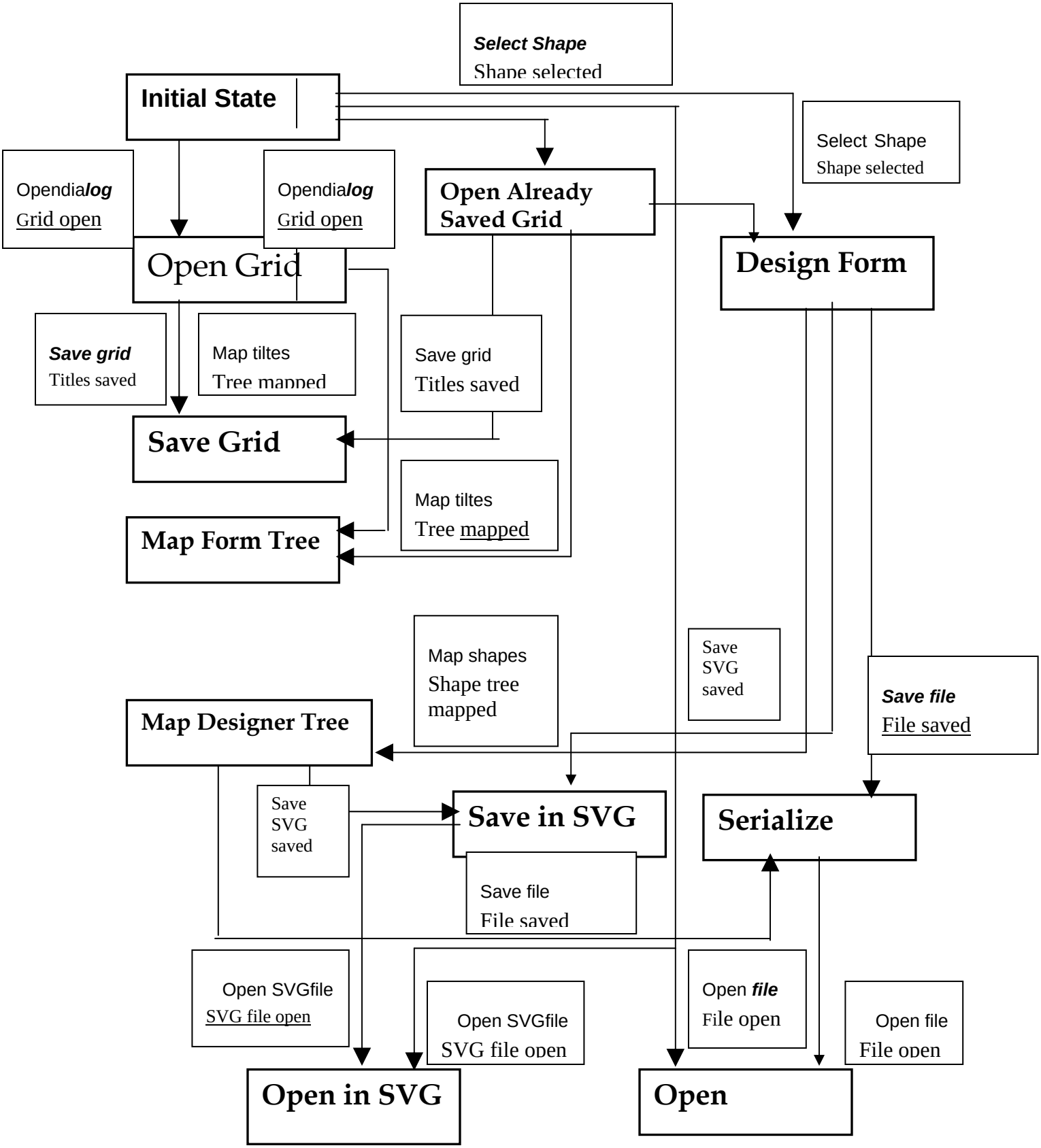


Section 13

Object Oriented Design

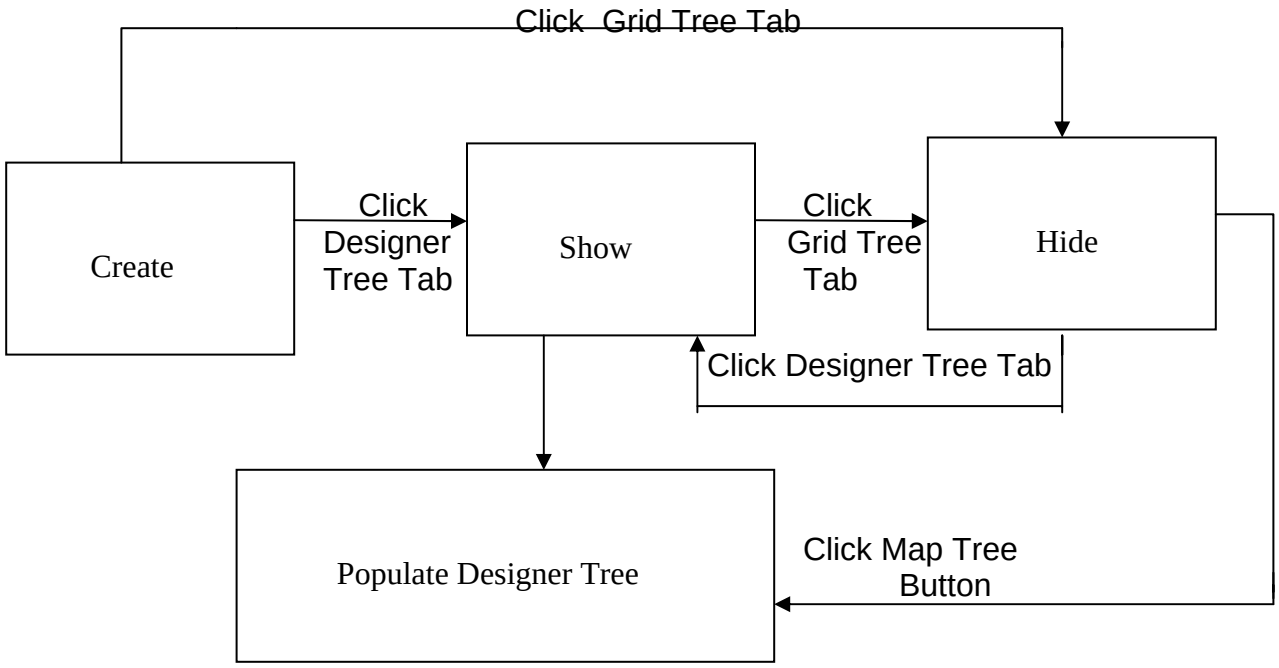
STATE CHARTS

State Chart for the System



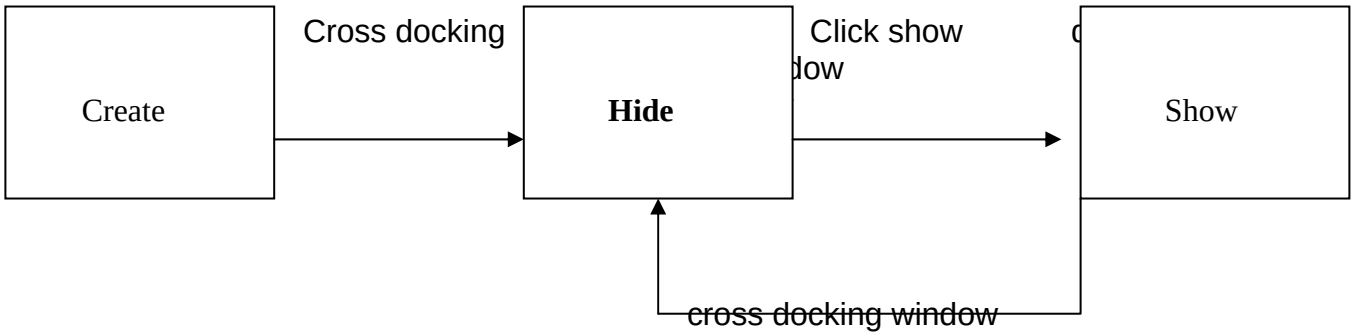
STATE CHARTS

CDesignerTree



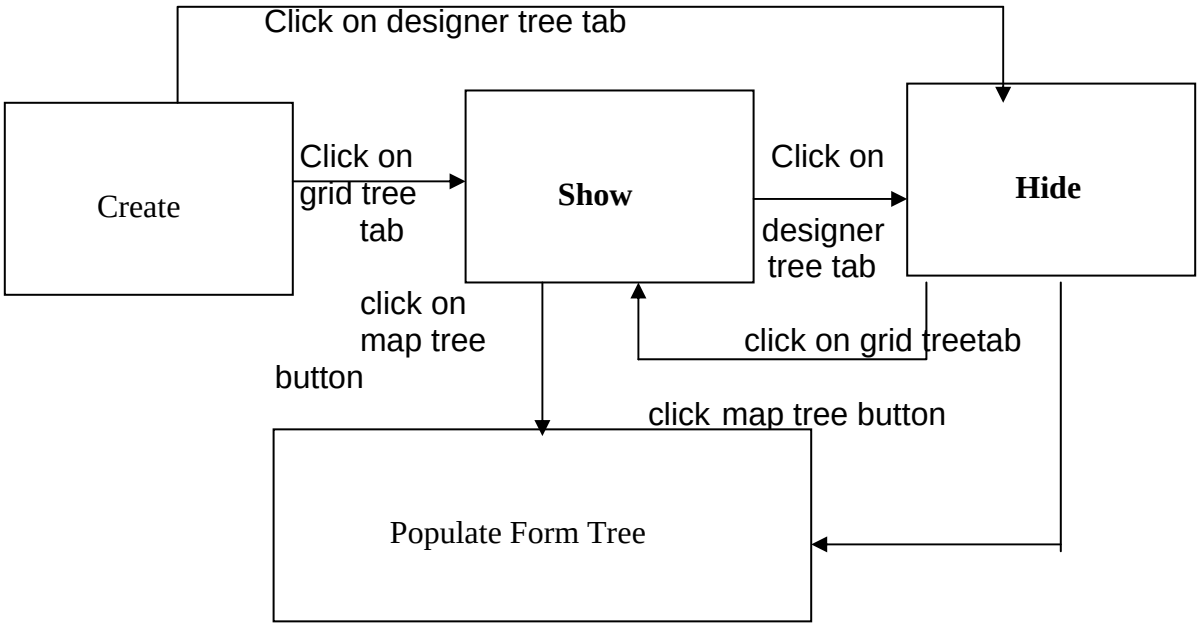
CDesignerTree is the class that creates Tree View of shapes drawn in the designer area .The initial state for that class is create state.When the application is started left docking window is created with designer tree created in it.From that state two staes can be achieved either hide by clicking Grid Tree tab button or we can show the tree by clicking designer tree tab.From both states of show or hide the user can move fom one to another .Also from both these states the user can populate the tree by the shapes drawn and can achieve populate designer tree state.

CDockingWindow



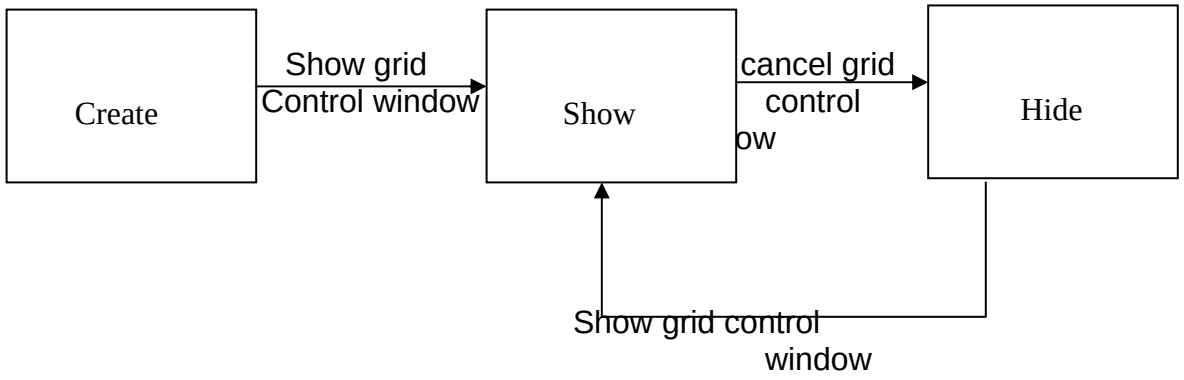
Cdockingindow is the class that creates left docking window .so initial state is “create” state. From that state the user can go to hide state by clicking on cross button. it can then go to show state by clicking Show docking window button.

CFormTree



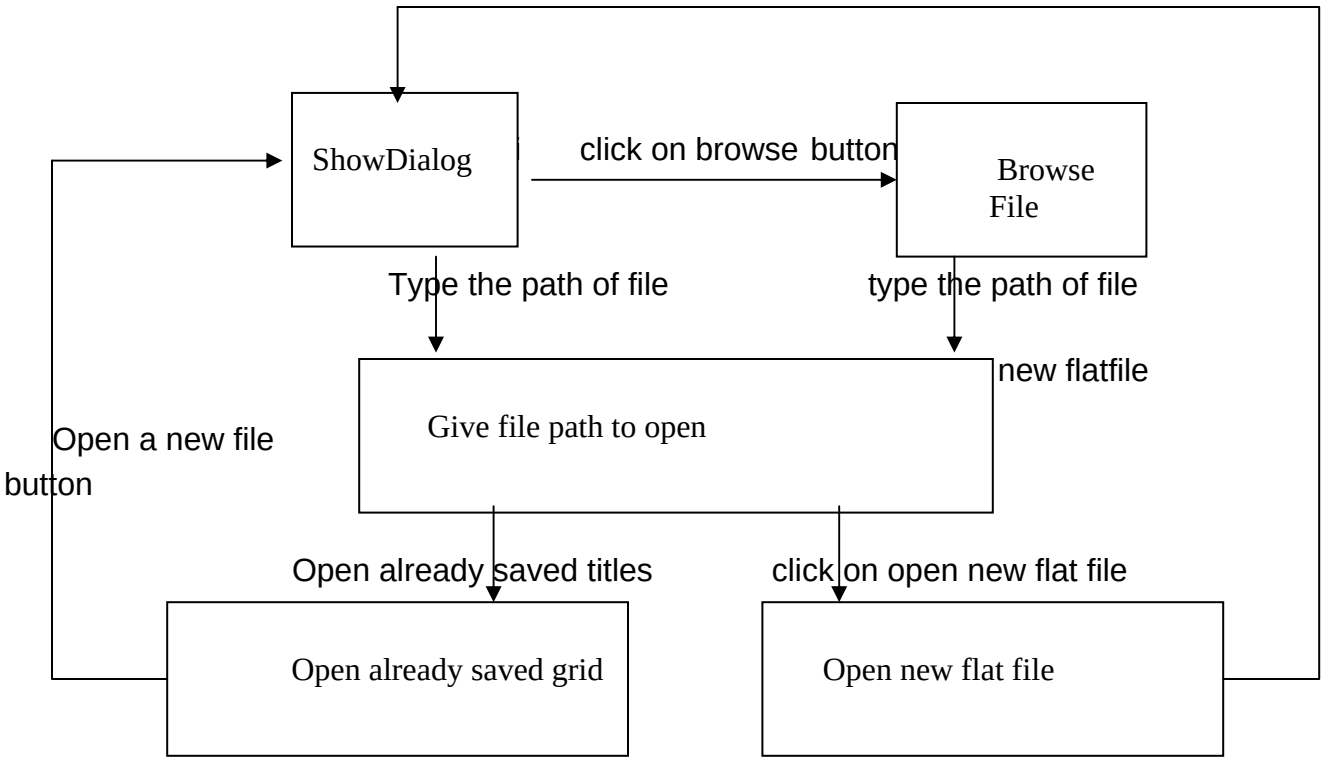
CForm Tree is the tree to map titles of grid .It is created in docking window initially .Using grid tree tab button that tree can be shown to move to show state or to hide state by clicking on designer tree button.Then it can move from hide to show state or vice versa by clicking on grid tree tab button and designer tree tab button respectively.Then from both of these states the tree can be populated and state will move to populate tree state by clicking populate tree button.

CGridWindow



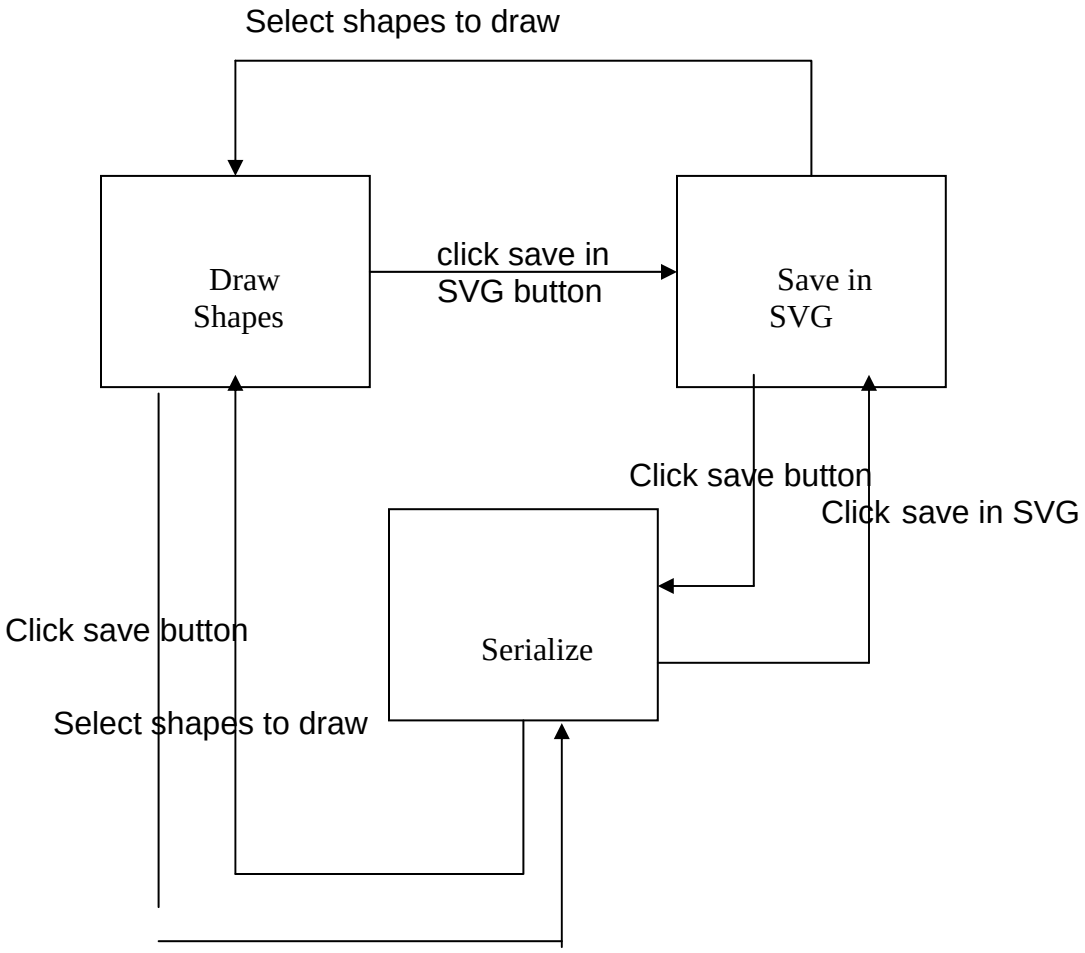
CGridWindow is the class that is used to create docking window for opening grid into it.You can show grid button to show grid window and you can hide it by cancelling the grid docking window.Then we can move to show state by clicking on show grid window button.

COpenDialog



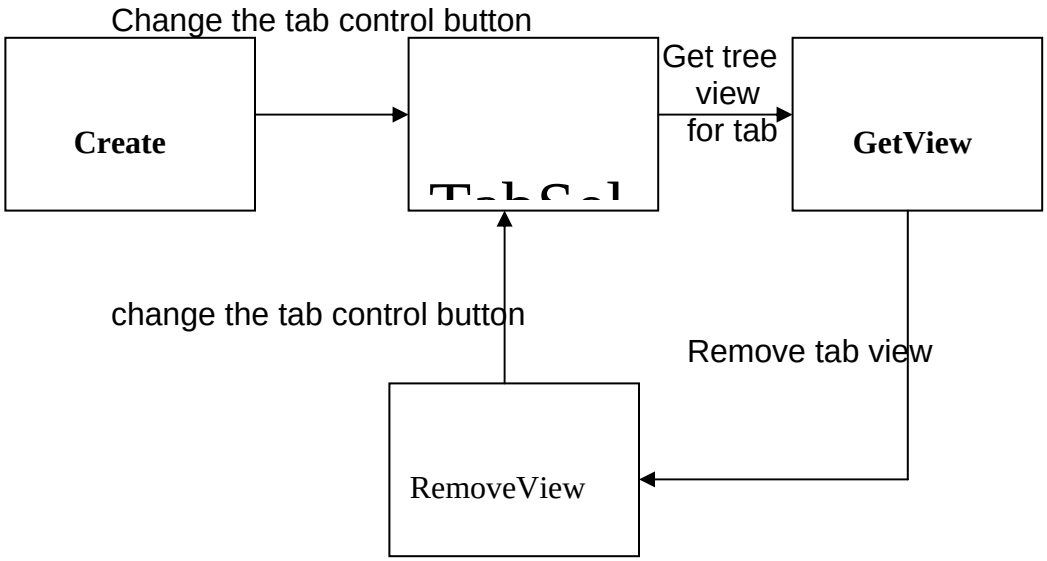
COpenDialog is used to create dialog box that is used to give the path of flat files to open it in a grid.The user can search the path of flat file to open it by using browse button to move to browse state or it can give the path of flat file manually and can move to Give flat file to open state.From this state the user can move to open already saved grid or it can open new flat file state by clicking on corresponding radio buttons.Then user can to initial state of show dialog by opening a new dialog.

CShapes



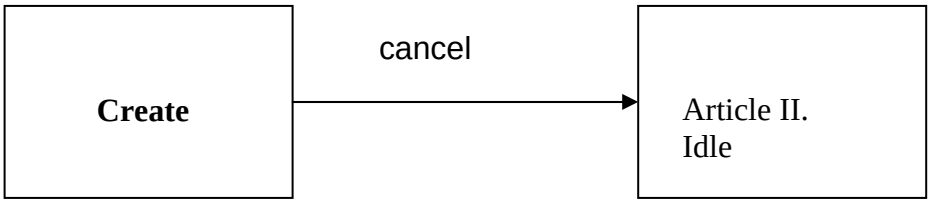
Cshapes is the class to drawshapes .Initial state is Draw Shapes state which is achieved by selecting shapes.The user can reach to serialize or save in SVG format clicking save button or save in SVG button.The user can also move between the states of serialize and save in SVG .From these states he can move to initial state.

CsizingTabCtrlBar



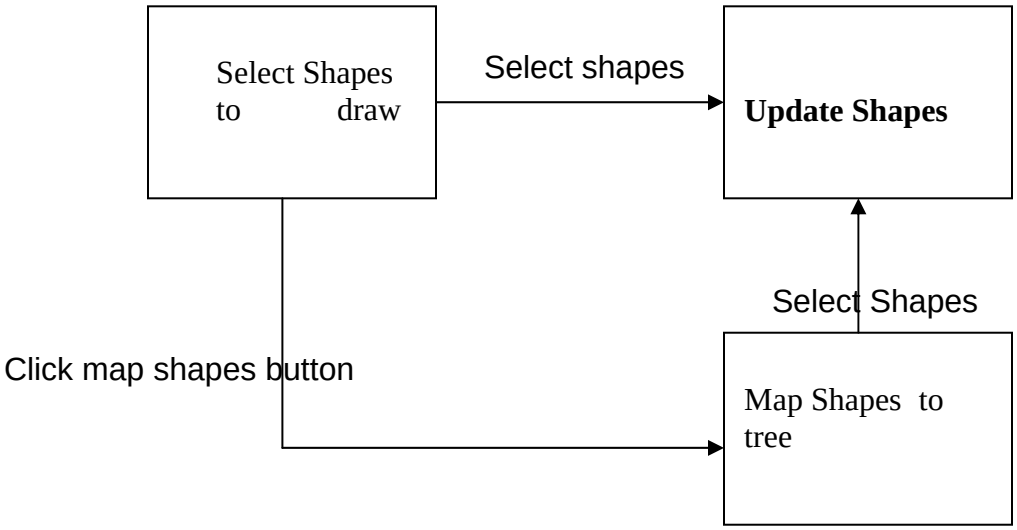
CsizingTabCtrlBar is used to create tab controls in docking window.After creating the tabs initially tab sel can be changed to achieve TabSelChange state.The user then moves to GetView state by the action of Get tree view of tab .and then it can remove the view of the tree as when one view will be created other will be removed.

CGrid



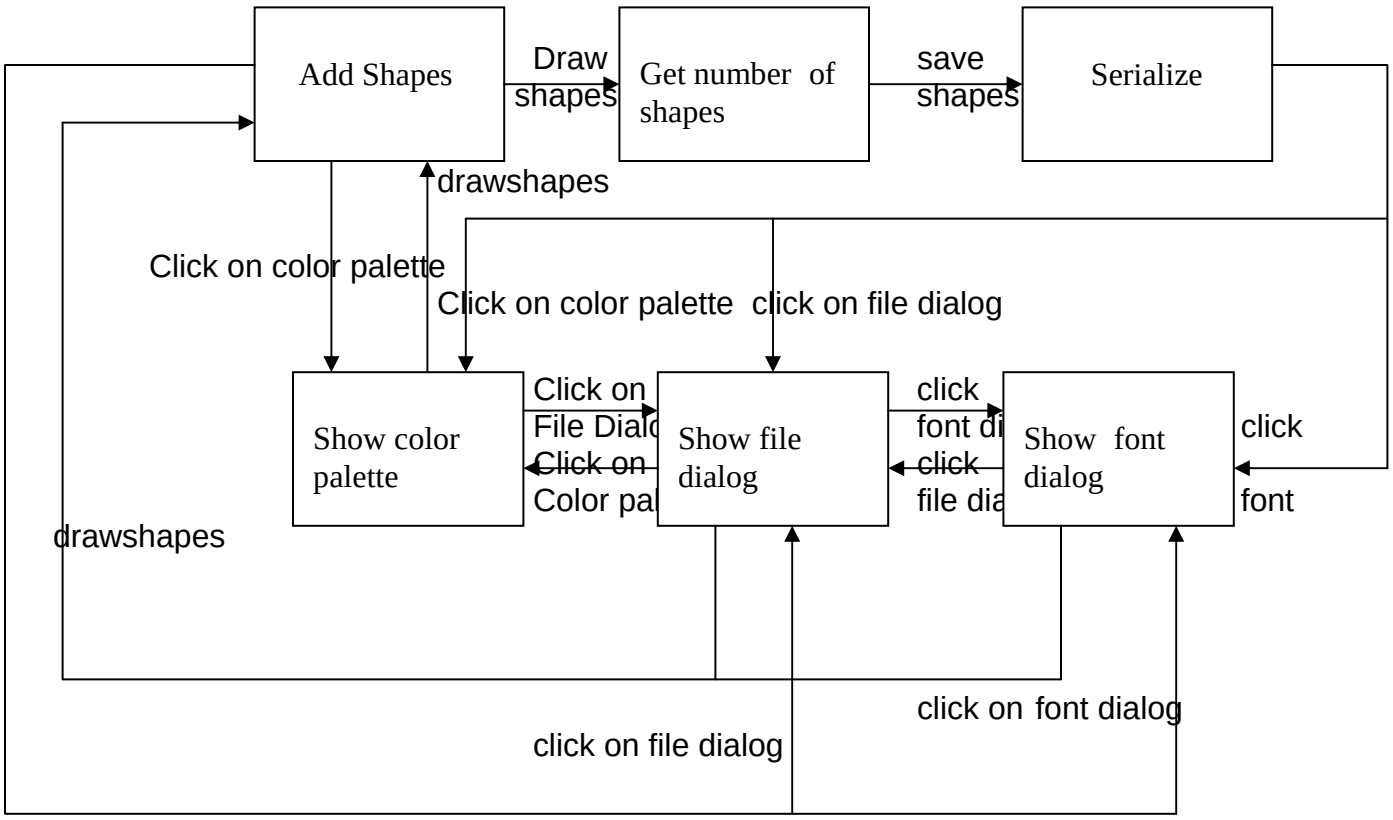
CGrid is used to create the basic layout of the grid that is opened in grid docking window .That grid is a MFC Grid Control.It has only two states .One it has create state and other state is Idle state which is attained after cancelling the grid.

CFormBuilderApp



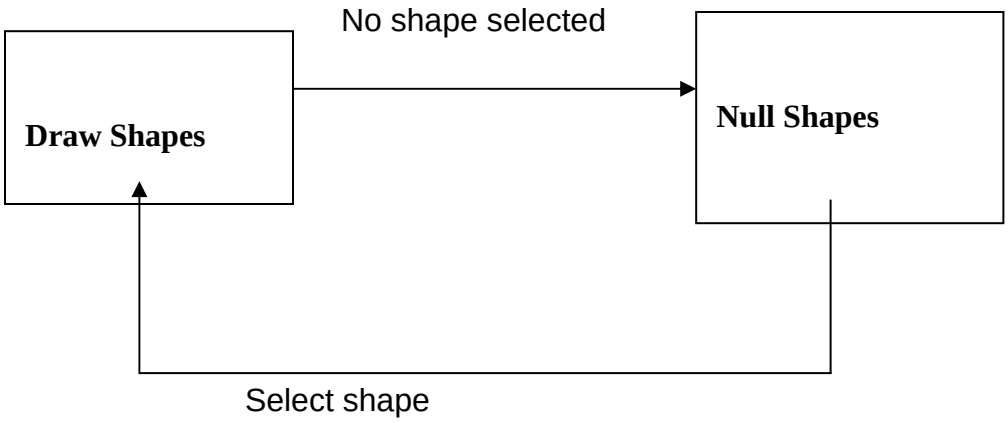
CformBuilderApp in an application class that is used to pass all the ID's of shapes.First you reach the state of Select shapes .That is designing face.Now you can easily map those shapes drawn using map shapes button.After mapping these shapes can be reselected that moves it to update Shapes.

CFormBuilderDoc



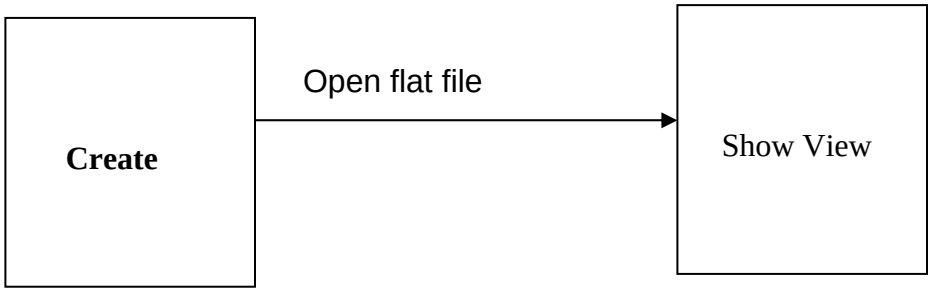
CFormbuilderdoc is the document class which is used to save the shapes drawn.whenever shapes are drawn shapes are added to the object array and state of Add Shape is achieved.After adding shapes their numbers can be get and state of Get number of shapes is achieved. From here shapescan ce saved and state of serialize is achieved.We can open the color palette and can reach the state of show color palette from Add shape state or serialize state.Show font dialog and show file dialog state can also be achieved from serialize state and from these states we can move to Add Shape state.

CFormBuilderView



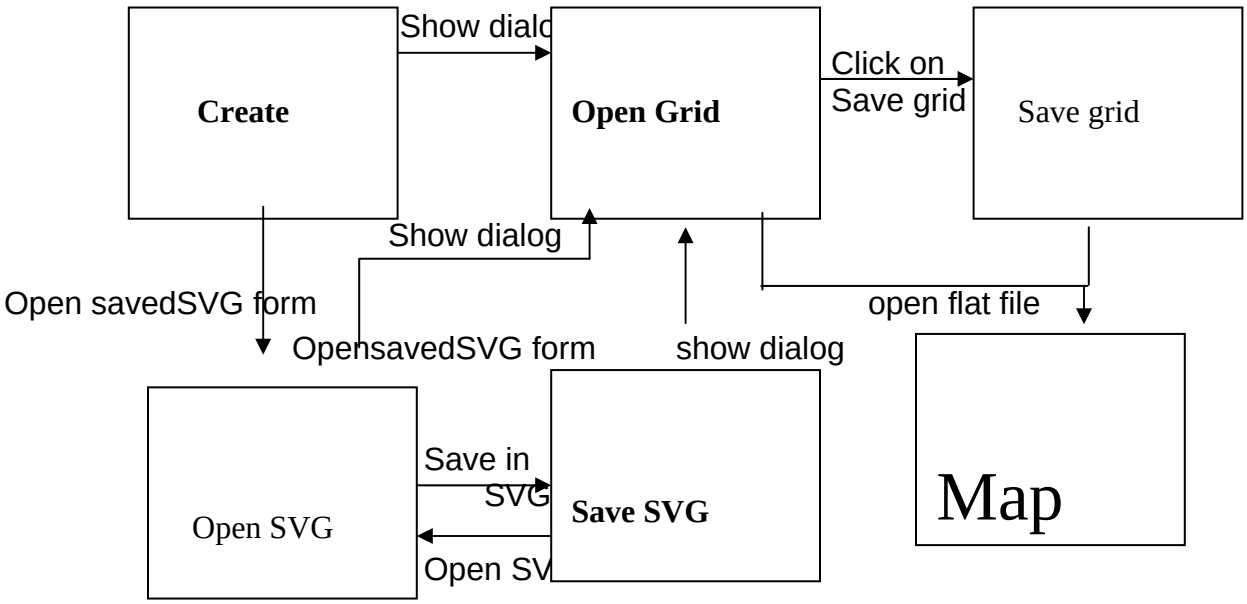
CformBuilderView is the class that is used to view the desgner area.here we have only two states.Either any shape is selected or user has selected no shape.

CGridView



CgridView class is used to view the MFC GridControl that contains flat files.Now it has only two states.In the first step it is created state that create view.Insecond step window inside the grid containg flat files is created.

CMainFrame



CmainFrame is the main class that handles the program when it is initiated.First state is the create state that creates the main layout od application like docking windows,toolbars and menubars.Now the user can move to two staes.Either to open grid state by opening dialog box or open SVG state by opening already saved document...From open grid statte user can move to two states either he can save the grid opened or it can map flat files opened onto tree.it can also move map flat file state after saving grid .Similarly after opening SVG document he can move to two states either he can open grid now or it can save after making changes in SVG saved documemnt.After saving he can then another document and can move to open SVG state.

SECTION 14

Object Oriented Design

DETAILED LOW LEVEL DESIGN

```
IF state=”Initial State”

{


action=”Open Grid”

do

{

edit grid();

rename grid();

assign titles();

map titles();

} while action=”Open grid”;

else if(action=”Open already saved grid”)

do

{

save grid();

edit grid();

title assigner();
```

```
title mapper();

} while action=”Open already saved grid”;

else if (action=”Design form”)

do

{

design form layout;

map objects on designer tree;

map titles on form;

serialize form in SVG;

serialize form in fbd;


} while(action=”Design form”)

}

IF state=”Map designer tree”

{

if(action=”Save in SVG ”)

{
```

SVG serializer();

}

else if(action=”Open in SVG”)

{

SVG opener();

}

else if(action=”Open”)

{

OpenDialog();

}

else if(action=”Serialize”)

{

serializer();

}

}

IF state=”Map Form Tree”

{

```
createFormTree();

maintainFormTree();

mapObjects();

}
```

Section 15

TEST CASES

Equivalence Classes and Boundary Value Analysis

1. **Equivalence class for File Opening**

Less than one character	Error
One Character	Acceptable
Between 1 and 20 characters	Acceptable
Twenty characters	Acceptable
More than 20	Truncated to 20 characters

2. **Equivalence class for File Saving**

Less than one character	Error
One Character	Acceptable
Between 1 and 20 characters	Acceptable
Twenty characters	Acceptable
More than 20	Truncated to 20 characters

3. Equivalence class for Bitmap Opening

Less than one character	Error
One Character	Acceptable
Between 1 and 20 characters	Acceptable
Twenty characters	Acceptable
More than 20	Acceptable

4. Equivalence class for Drawing Text

Less than one character	Acceptable
One Character	Acceptable
Between 1 and 20 characters	Acceptable
Twenty characters	Acceptable
More than 20	Acceptable

Functional Analysis

1. Open the designer.
2. Close the designer.
3. Open the grid.
4. Close the grid
5. Open a flat file.
6. Activate the tree docking window.
7. Deactivate the tree docking window.
8. Activate the grid docking window.
9. Deactivate the grid docking window
10. Assign titles to the flat file.
11. Map titles to grid tree.
12. Delete titles in the flat file opened in the grid control.
13. Change titles in the flat file opened in the grid control.
14. Activate the grid tree view in the tree docking window.
15. Deactivate the grid tree view in the tree docking window.
16. Activate the designer tree view in the tree docking window.
17. Deactivate the designer tree view in the tree docking window.
18. Activate the shape toolbar.
19. Deactivate the shape toolbar.
20. Dock the shape toolbar.
21. Resize the shape toolbar.
22. Select shapes from shape toolbar.
23. Select line from shape toolbar.
24. Draw line on the designer area.
25. Select rectangle from shape toolbar.
26. Draw rectangle on the designer area.
27. Select transparent circle from the shape toolbar.
28. Draw transparent circle on the designer area.
29. Select transparent rectangle from the shape toolbar.

30. Draw transparent rectangle on the designer area.
31. Select filled circle from the shape toolbar.
32. Draw filled circle on the designer area.
33. Select filled rectangle from the shape toolbar.
34. Draw filled rectangle on the designer area.
35. Select pencil from the shape toolbar.
36. Draw with pencil on the designer area.
37. Select polygon from the shape toolbar.
38. Draw polygon on the designer area.
39. Select gray circle from the shape toolbar.
40. Draw gray circle on the designer area.
41. Select light gray circle from the shape toolbar.
42. Draw light gray circle on the designer area.
43. Select dark gray circle from the shape toolbar.
44. Draw dark gray circle on the designer area.
45. Select gray rectangle from the shape toolbar.
46. Draw gray rectangle on the designer area.
47. Select light gray rectangle from the shape toolbar.
48. Draw light gray rectangle on the designer area.
49. Select dark gray rectangle from the shape toolbar.
50. Draw dark gray rectangle on the designer area.
51. Select color tool from the shape toolbar.
52. Change the color of the selected shape.
53. Change the color of the filled circle.
54. Change the color of the filled rectangle.
55. Change the color of the line.
56. Change the color of the pencil tool.
57. Select the thickness increment tool.
58. Change the thickness of the line
59. Select the thickness decrement tool
60. Change the thickness of the line.

61. Copy grid tree items on the designer area.
62. Map shapes on the designer tree.
63. Expand grid tree.
64. Populate grid tree.
65. Fold the grid tree.
66. Expand the designer tree.
67. Populate designer tree.
68. Fold the designer tree.
69. Save the grid tree.
70. Save the designer tree.
71. Save the file in fbd format.
72. Save the file in SVG format.
73. Open the file in fbd format.
74. Open the SVG file.
75. Hide the grid.
76. Unhide the grid.
77. Close the tree docking window.
78. Close the grid docking window.
79. Resize the grid and tree docking windows.
80. Close the application

Section 16

USER MANUAL

This software is to be used by someone who wants to design a form with the help of a flat file or without the help of flat file. It can also be used by someone who just wants to assign titles to a flat file and then save it for later use. User can save the designed forms in the format of this program i.e. *.fbd format and international standard of format i.e. SVG format so that the form can be displayed on the internet explorer browser using a plug-in for internet explorer, known as SVGView. Oldest version of the browser on which it can be seen is internet explorer 6.0 or Netscape 6.0.

First of all, when the application is run, a splash screen appears as shown in Figure 16.1 and then disappears itself after a few moments. Then the application looks like the one shown in Figure 16.2.

Now in the above figure, there is a designer in the middle of the application with white background; there is a window on the left side containing the tab view with which two tree views are associated. In the right tab titled as “Grid Tree”, titles of the flat file are to be mapped. In the left tab titled as “Designer Tree”, shapes drawn and their attributes are to be mapped.

MainFrame ToolBar

Now if you see on the top of the application, there is a toolbar docked there. This is the mainframe toolbar in which different options are there. Starting from left to right in the mainframe toolbar, first of all we see a button with a page on it. This button is there to open a new document in this application to work with.

After the first button, there is a separator which is then followed by three buttons which give options of opening files. The first one, on which an arrow is coming from a diskette into a grid shows that this button is used to open a flat file are already saved grid titles into a grid. If, in an application, a grid is opened, no other grid can be opened unless the application is run again. If you click on this button except for the first time, it will open the same grid which it opened after parsing the first chosen flat

file or the file containing grid titles. The changes will remain there if changed after opening the grid.

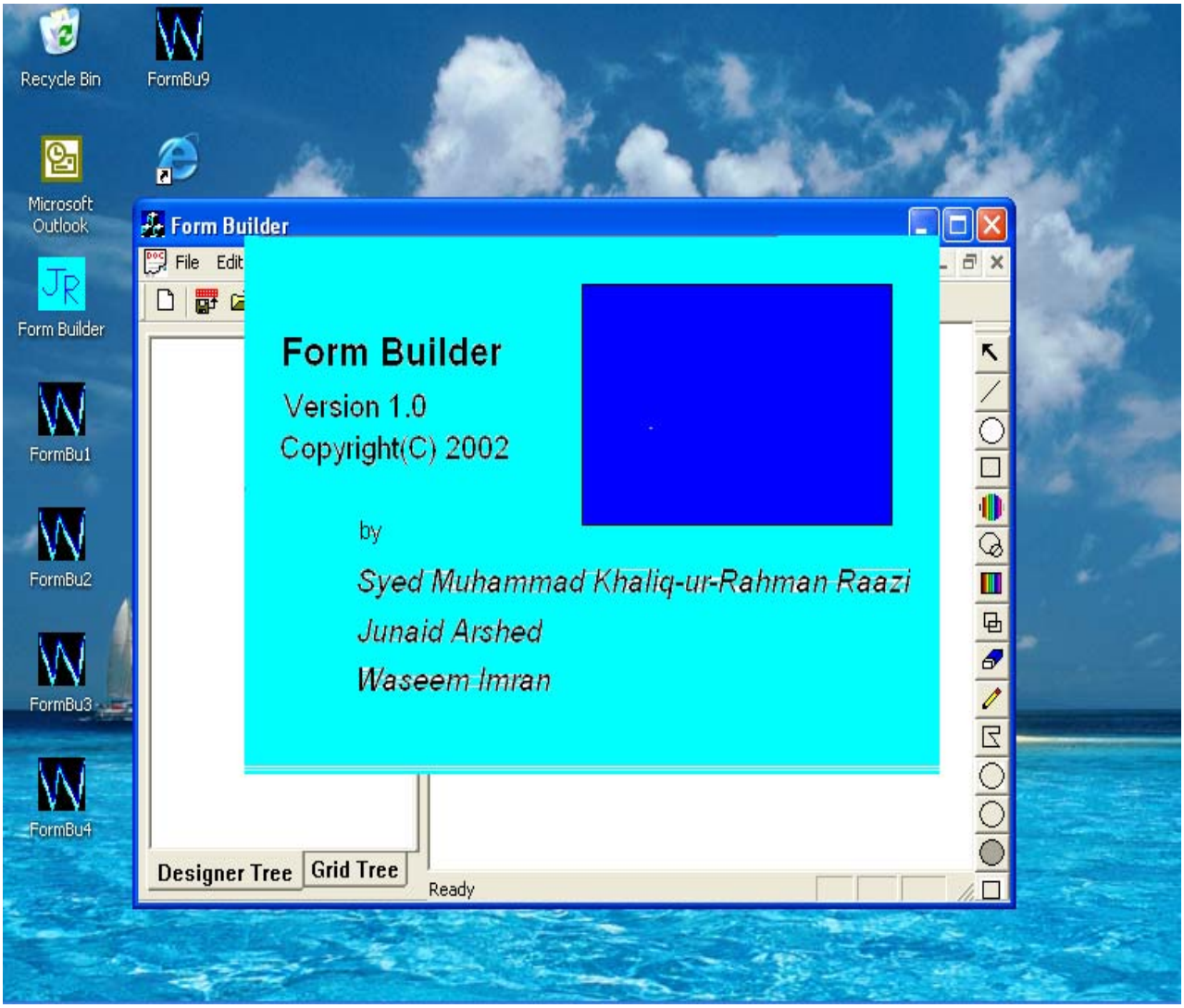


Figure 16.1

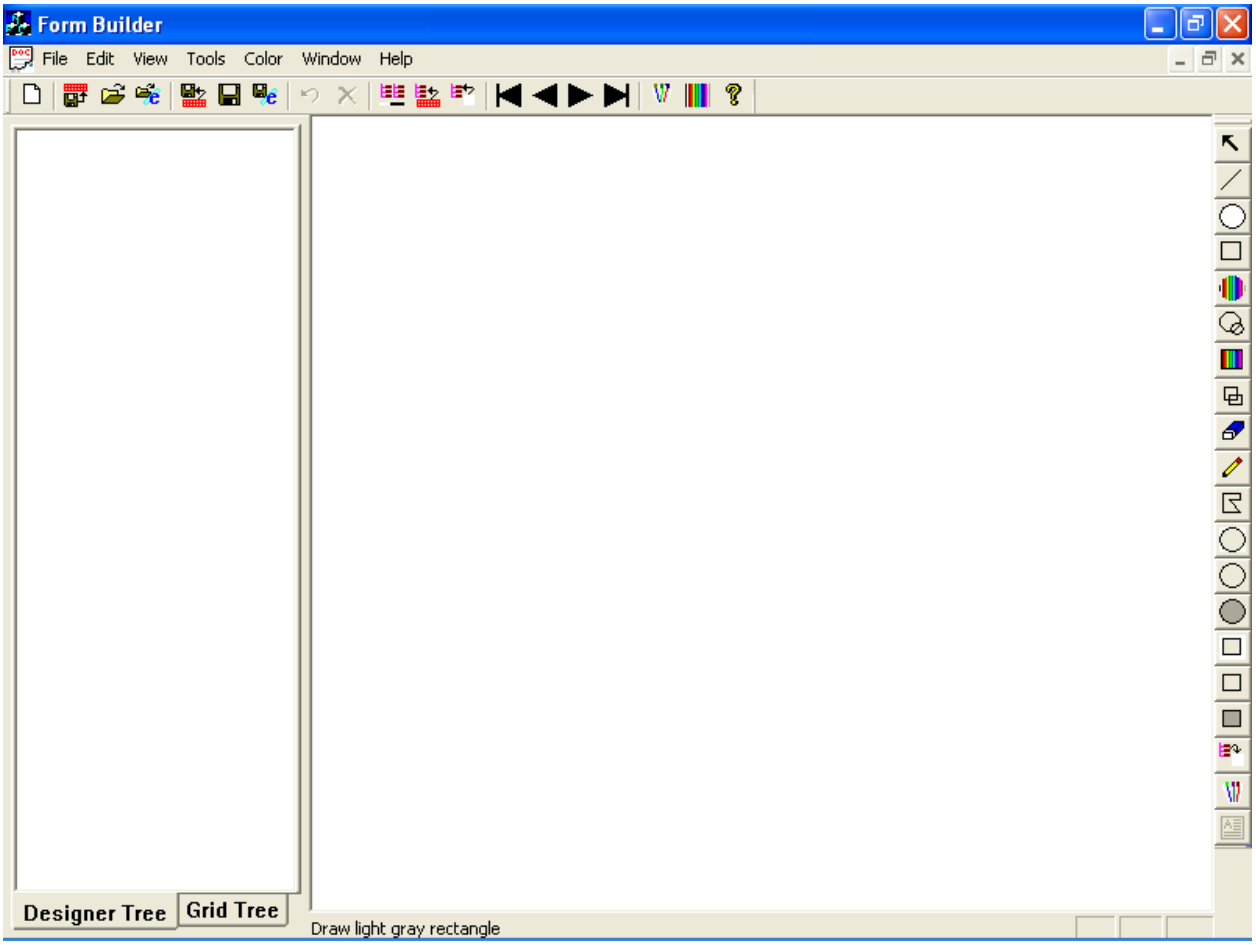

Form Builder
File Edit View Tools Color Window Help
Designer Tree
Grid Tree
Draw light gray rectangle

Figure 16.2

The third button is to open a normally saved form designer into a new document having an extension *.fbd. The fourth button is there to open an already saved form in SVG format in a new document in the designer area.

There is another separator after the fourth button, and then there are three buttons in series. First button in this series gives the option to the user to save the grid titles assigned to the flat file document opened in the grid. It will be saved in a file which can be accessed and opened in this document. Second option in this series is a button to save the current document in the format of this Form Builder i.e *.fbd format. The third button is the option to save the current active document in the SVG format.

After that, there is a separator and then there are two options, first one is “undo” option in which last drawn shape is deleted. The second one is “clear all” option in which the currently active form will be deleted from the designer area. Then again there is a separator.

Then there are three buttons. First one is to show the window initially docked on the left if it has been closed. It will open the window where it was closed because it can also fly. Second button is an option for the user to map the titles, given to the flat file onto the grid, to the tree view on the tab named “Grid tree” on the tab control. Then there is an option to map the shapes drawn and their attributes on the tree view on the tab named “Designer tree”.

Then there is another separator followed by four buttons, which are used to manipulate the pen width for drawing the shapes. If first button is pressed, the pen width will become the minimum, if the second one is pressed, the width decreases by one, if the third one is pressed, the width increases by one and if the fourth one is pressed, the width will become maximum.

There is another separator followed by three buttons. The first button in this case chooses a bitmap file to be drawn onto the designer. The second option chooses the color with which the shapes will be drawn onto the designer area. The third button shows the information about the name and version of the software.

MenuBars

All the menu bars have options also present in the toolbars except the “view” menubar. It contains three options as shown in Figure 16.3

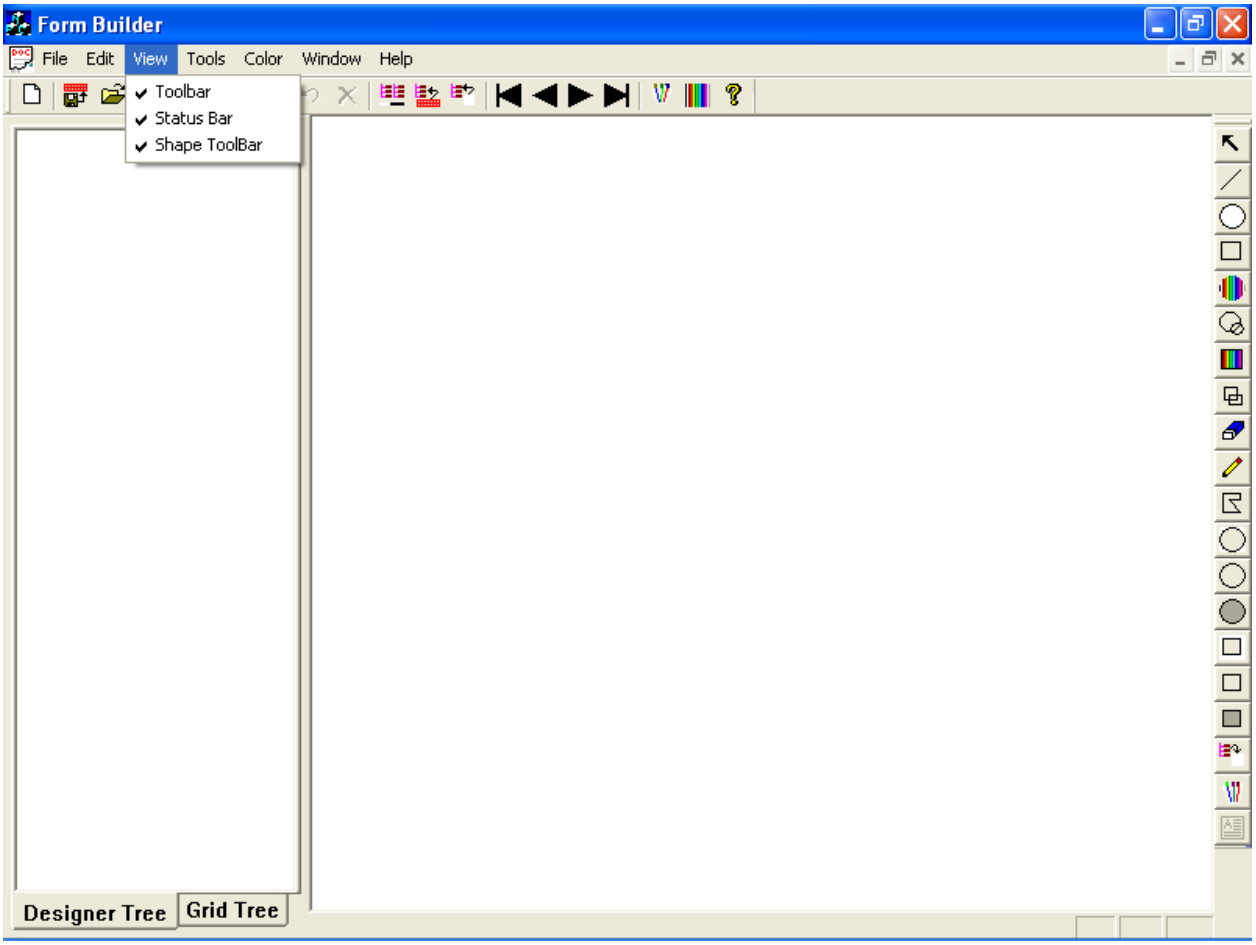


Figure 16.3

There are three menus in it named as “Toolbar”, “Status Bar” and “Shape toolbar”. These are all checkbox menu items. If the first one is checked, the mainframe toolbar is shown, otherwise it is hidden. If the second option is checked, it will show the status bar, otherwise it will not show the status bar. If the third option is checked, it will show the shape toolbar, otherwise it will not show the shape toolbar.

Shape ToolBar

If you see at the right of the application, we have a toolbar on which shapes are selected which are to be drawn. Starting from top, first one is an arrow which is null shape. If this option is selected, no shape is selected actually. Second one is the “line” option. If this button is selected, user can draw a line from the position on which he pressed the left mouse button down to the place where he released it. Similarly all circles and rectangles are drawn. Third option is a circle with white background. Fourth option is rectangle with white background. Fifth option is colored ellipse; the sixth one is transparent ellipse. Seventh option is colored rectangle, eighth option is transparent rectangle. Ninth option is rubber and tenth option is for free hand drawing. In both the cases of rubber and free hand drawing, whenever the mouse is moved when pressed, a line is drawn between the starting point of the movement and the ending point of the mouse movement. Line is of selected width and color in case of free hand drawing and of white color in case of rubber. After that, there are options to draw light gray ellipse, gray ellipse, dark gray ellipse, light gray rectangle, gray rectangle and dark gray rectangle respectively. They can be drawn just as other ellipses and rectangles with light gray, gray or dark gray backgrounds. There is also a polygon option to draw a line to the place where clicked, from the previous point to which the pervious shape was drawn.

There are three other options. The third last option is to select a tree item from the grid tree and when user clicks on the designer, the selected tree item will be pasted on the designer. Next option is of the bitmaps. If this option is selected and a bitmap is already selected in the Mainframe toolbar, that bitmap will be drawn from the point where the left mouse button went down to the point where it is released. If the mouse is clicked on the designer, the bitmap will be pasted onto the designer, top left of the figure being the point where the mouse was clicked. The last option in this toolbar is of an edit control. This edit control is formed on the designer between the points where the left mouse button was pressed to the point where it was released.

Then mouse can be clicked in this area to write the text. The edit control is not shown; only the cursor is shown when clicked inside the edit control as shown in the figure 16.4, and when the mouse button will be clicked outside this area, the edit box will disappear, drawing its text on its starting position.

Now coming back onto the flat files, if you choose the option of opening flat file for the first time, a dialog box will be shown which gives the option to either browse for a file or to enter a path into a text box as shown in Figure 16.5. Now you have two options over here. Either a flat file can be opened in the even rows or already saved grid titles can be opened in the odd rows. Opened flat file in the bottom of the application is shown in figure 16.6 and already saved grid titles which are opened are also shown in figure 16.7. It can also be like this that a person opens a flat file, gives it titles and then map them onto the grid tree. Mapped titles onto the grid tree are shown in figure 16.8. Note that the first column of every row is made the root and the subsequent columns the same row are made its leaves as in the flat files, the first field of a record always contains the record identity. Then the grid window is closed and the form is designed in the designer.

When the design of the forms is complete, the shapes drawn can be mapped onto the designer tree with shape number being the root and attributes of the shapes being the leaves. Attributes of a shape depend upon the shape which is mapped or drawn. The shapes and their mapping into the designer tree are shown in figure 16.9. Now from this point onwards, we can save our work in *.fbd format and SVG format. Both the toolbars i.e. Main frame tool bar and the shape toolbar are docking, the window on the left and the one at the bottom containing grid are docking but they can be flied at any position even outside the mainframe as shown in figure 16.10.

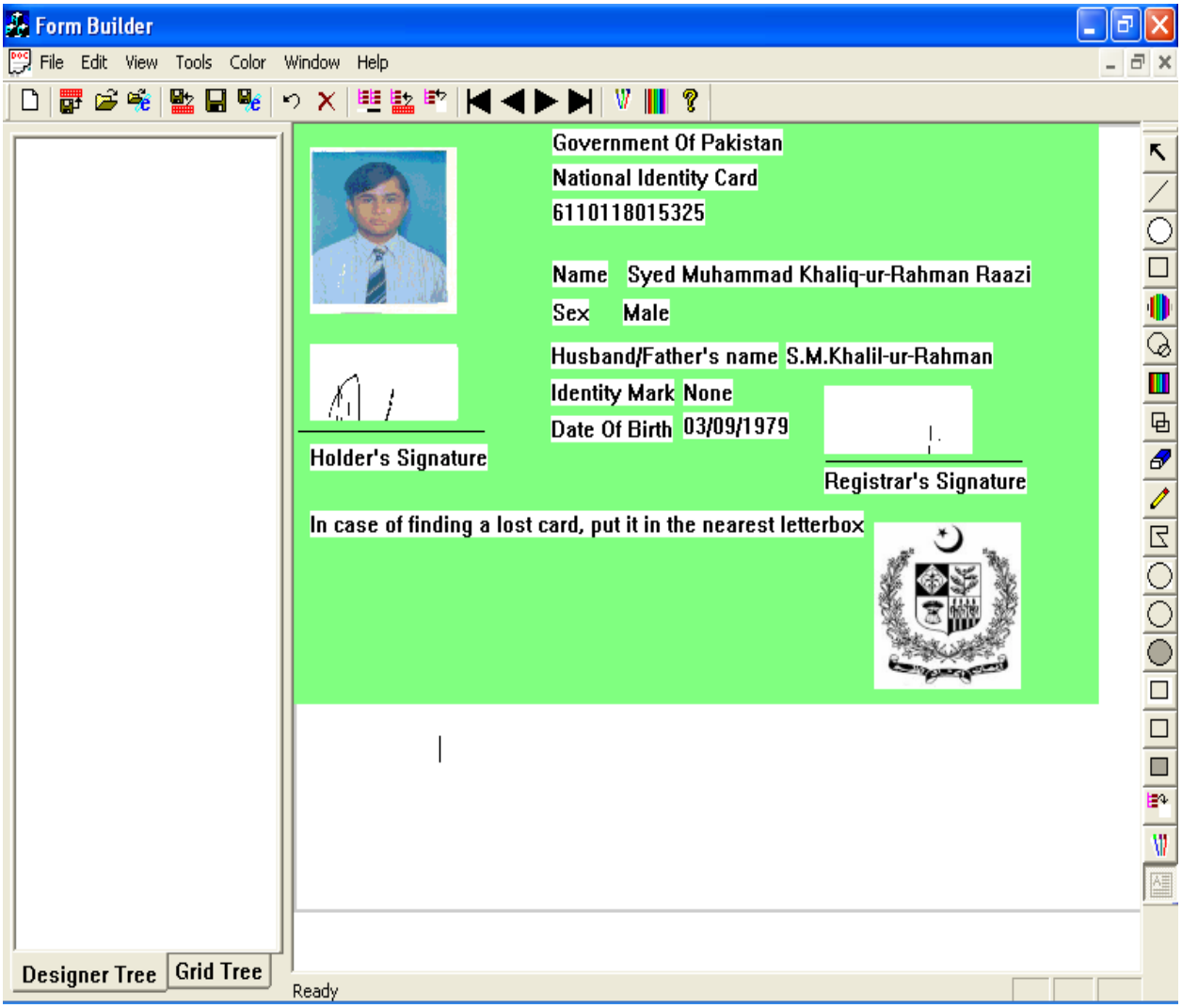

Form Builder
File Edit View Tools Color Window Help
Government Of Pakistan
National Identity Card
6110118015325
Name Syed Muhammad Khaliq-ur-Rahman Raazi
Sex Male
Husband/Father's name S.M.Khalil-ur-Rahman
Identity Mark None
Date Of Birth 03/09/1979
Holder's Signature
Registrar's Signature
In case of finding a lost card, put it in the nearest letterbox
Designer Tree
Grid Tree
Ready

Figure 16.4

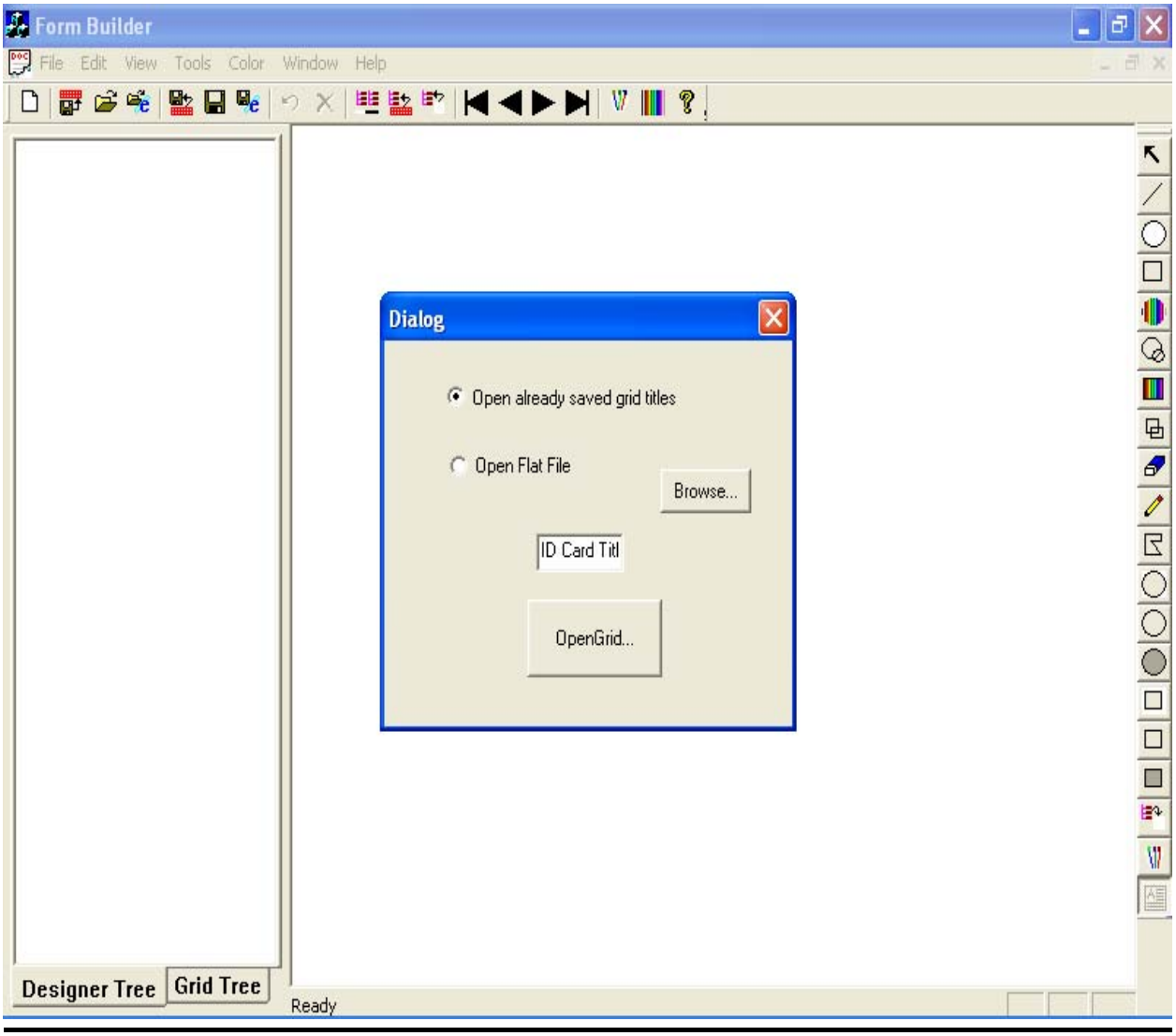
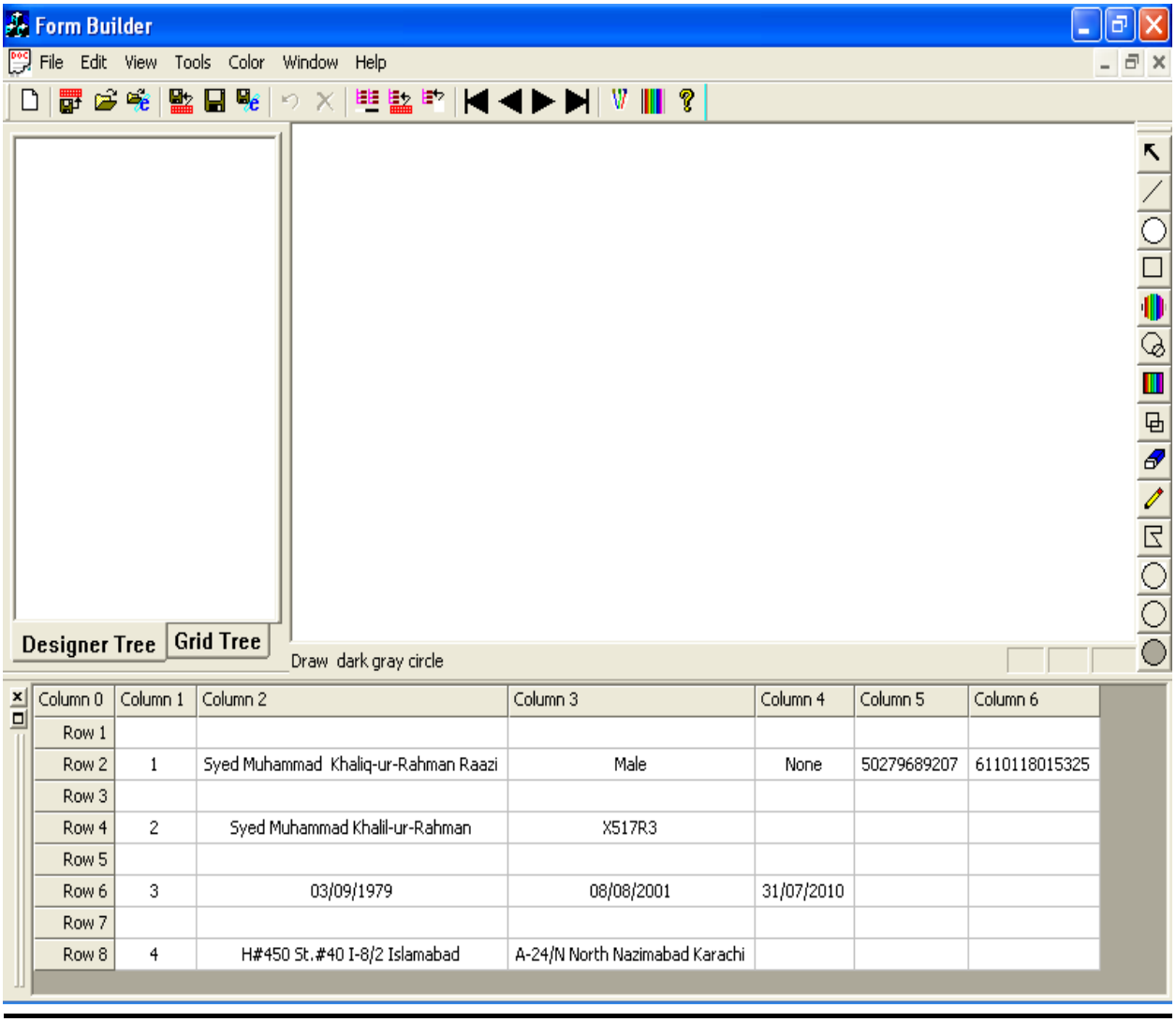

Form Builder
File Edit View Tools Color Window Help
Dialog
Open already saved grid titles
Open Flat File
Browse...
ID Card Titl
OpenGrid...
Designer Tree
Grid Tree
Ready

Figure 16.5



Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Row 1						
Row 2	1	Syed Muhammad Khaliq-ur-Rahman Raazi	Male	None	50279689207	6110118015325
Row 3						
Row 4	2	Syed Muhammad Khalil-ur-Rahman	X517R3			
Row 5						
Row 6	3	03/09/1979	08/08/2001	31/07/2010		
Row 7						
Row 8	4	H#450 St.#40 I-8/2 Islamabad	A-24/N North Nazimabad Karachi			

Figure 16.6

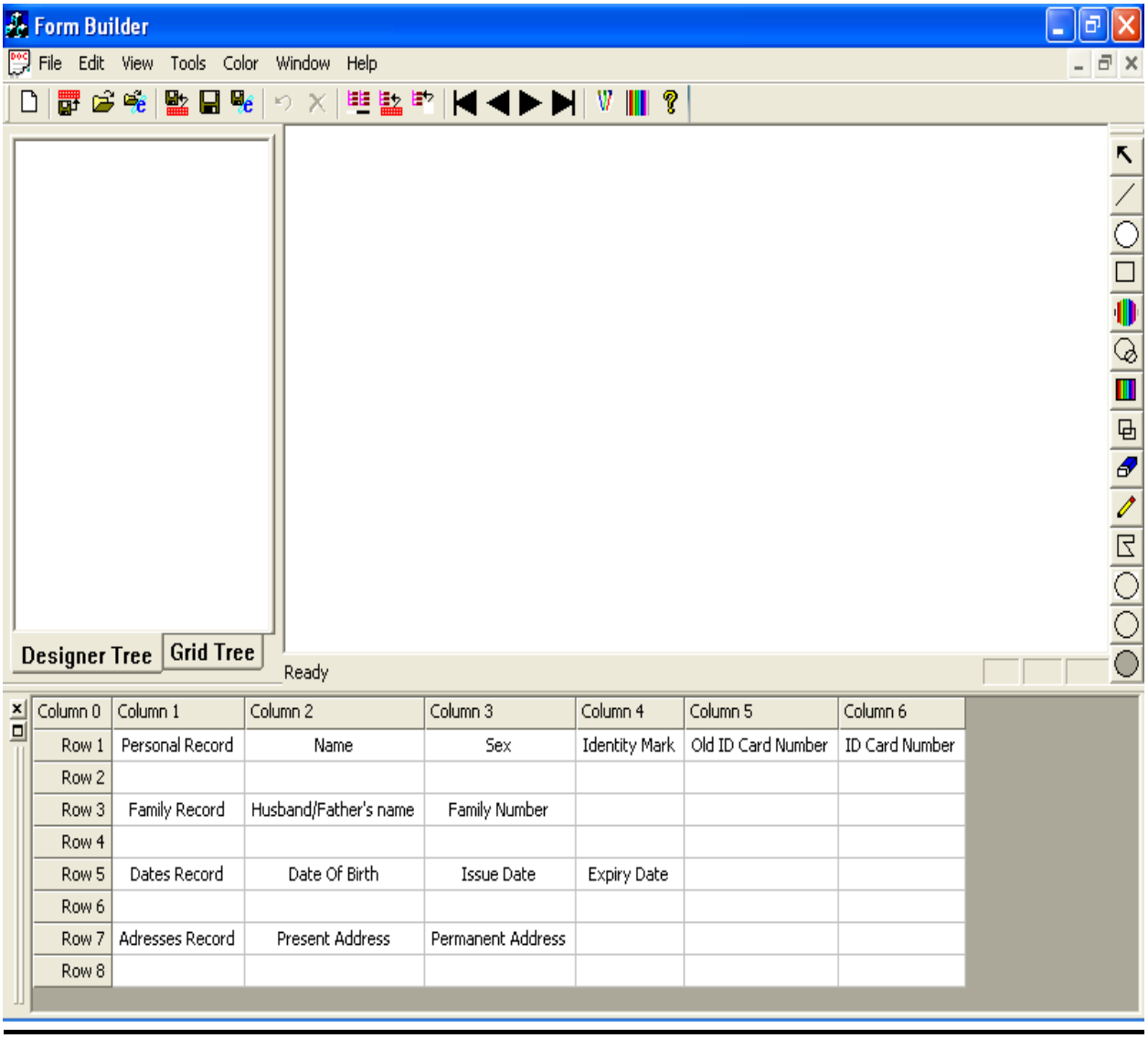

Form Builder
File Edit View Tools Color Window Help
Designer Tree
Grid Tree
Ready
Column 0 Column 1 Column 2 Column 3 Column 4 Column 5 Column 6
Row 1 Personal Record Name Sex Identity Mark Old ID Card Number ID Card Number
Row 2
Row 3 Family Record Husband/Father's name Family Number
Row 4
Row 5 Dates Record Date Of Birth Issue Date Expiry Date
Row 6
Row 7 Adresses Record Present Address Permanent Address
Row 8

Figure 16.7

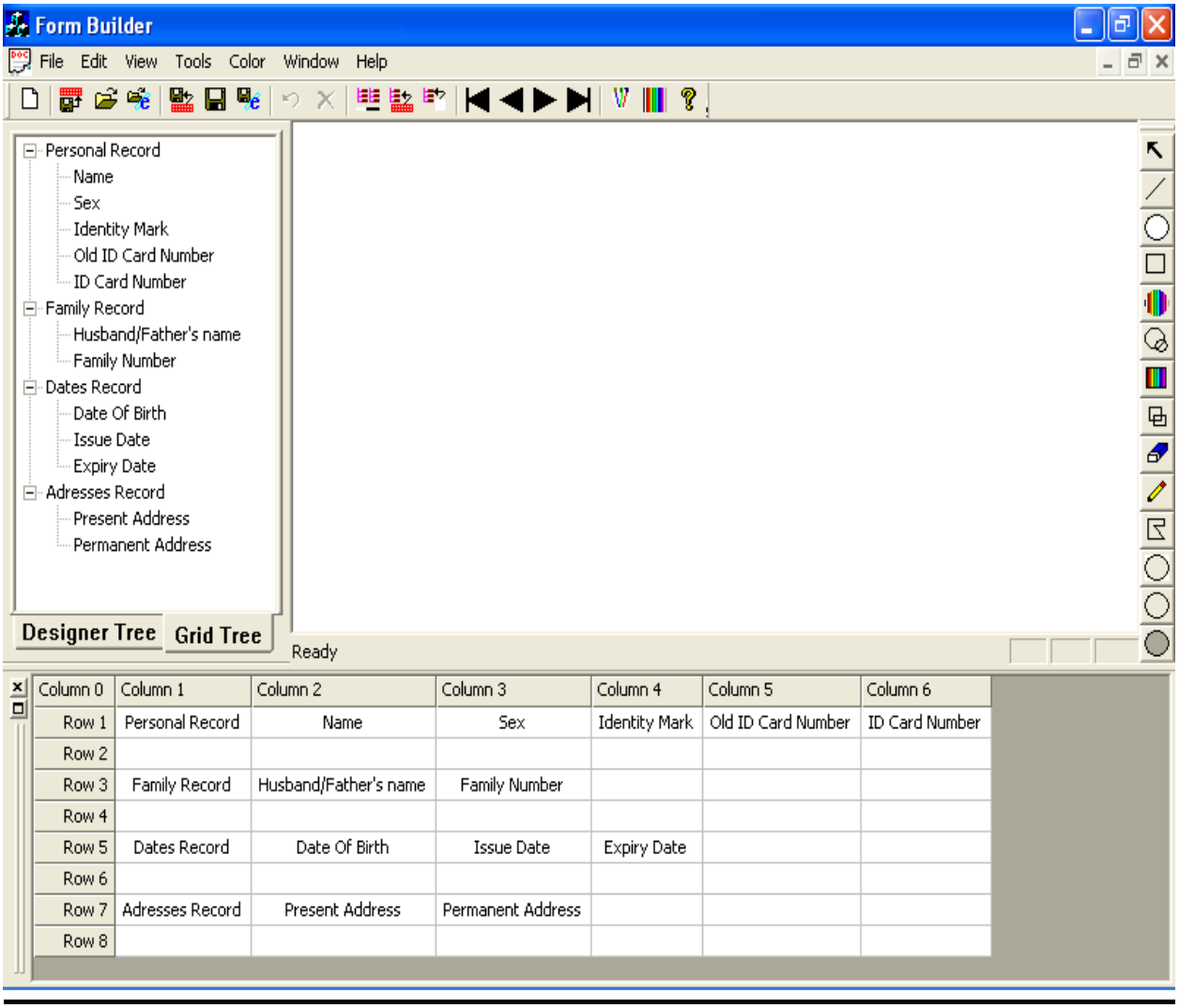

Form Builder
File Edit View Tools Color Window Help
Personal Record
Name
Sex
Identity Mark
Old ID Card Number
ID Card Number
Family Record
Husband/Father's name
Family Number
Dates Record
Date Of Birth
Issue Date
Expiry Date
Adresses Record
Present Address
Permanent Address
Designer Tree
Grid Tree
Ready
Column 0 Column 1 Column 2 Column 3 Column 4 Column 5 Column 6
Row 1 Personal Record Name Sex Identity Mark Old ID Card Number ID Card Number
Row 2
Row 3 Family Record Husband/Father's name Family Number
Row 4
Row 5 Dates Record Date Of Birth Issue Date Expiry Date
Row 6
Row 7 Adresses Record Present Address Permanent Address
Row 8

Figure 16.8

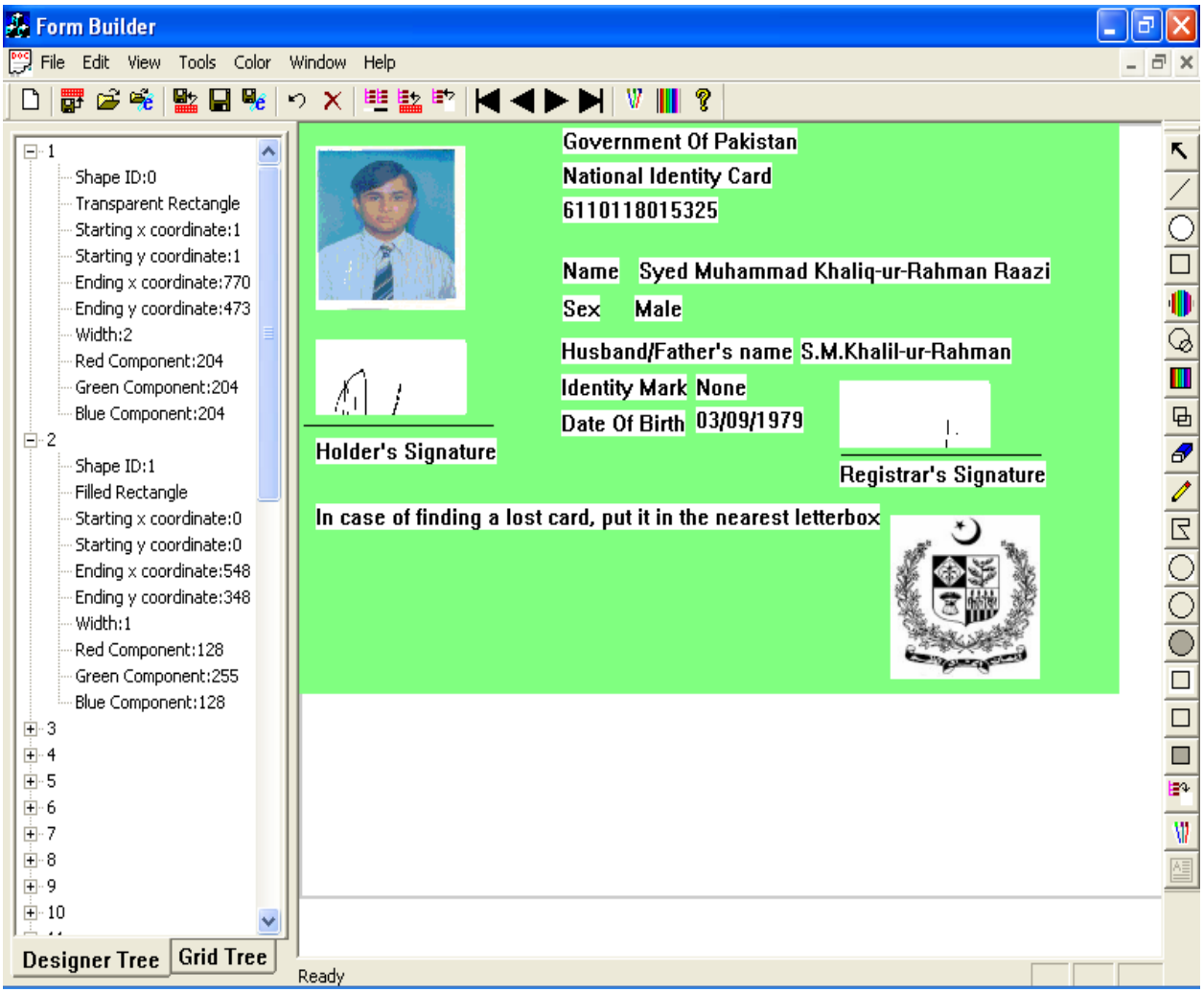

Form Builder
File Edit View Tools Color Window Help
1
Shape ID:0
Transparent Rectangle
Starting x coordinate:1
Starting y coordinate:1
Ending x coordinate:770
Ending y coordinate:473
Width:2
Red Component:204
Green Component:204
Blue Component:204
2
Shape ID:1
Filled Rectangle
Starting x coordinate:0
Starting y coordinate:0
Ending x coordinate:548
Ending y coordinate:348
Width:1
Red Component:128
Green Component:255
Blue Component:128
3
4
5
6
7
8
9
10
Designer Tree
Grid Tree
Government Of Pakistan
National Identity Card
6110118015325
Name Syed Muhammad Khaliq-ur-Rahman Raazi
Sex Male
Husband/Father's name S.M.Khalil-ur-Rahman
Identity Mark None
Date Of Birth 03/09/1979
Holder's Signature
Registrar's Signature
In case of finding a lost card, put it in the nearest letterbox
Ready

Figure 16.9

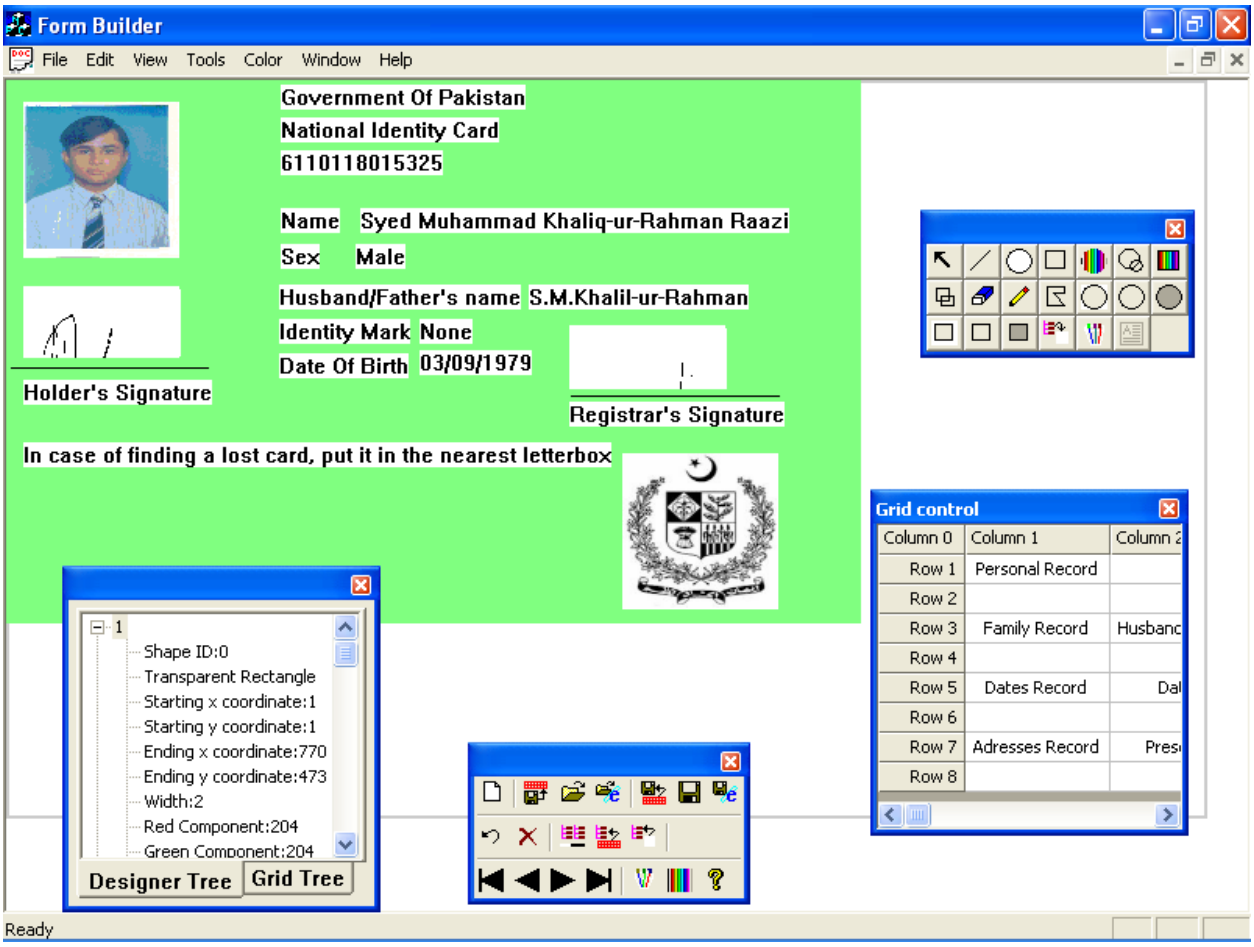

Form Builder
File Edit View Tools Color Window Help
Government Of Pakistan
National Identity Card
6110118015325
Name Syed Muhammad Khaliq-ur-Rahman Raazi
Sex Male
Husband/Father's name S.M.Khalil-ur-Rahman
Identity Mark None
Date Of Birth 03/09/1979
Holder's Signature
Registrar's Signature
In case of finding a lost card, put it in the nearest letterbox
Grid control
Column 0
Column 1
Row 1
Personal Record
Row 2
Row 3
Family Record
Row 4
Row 5
Dates Record
Row 6
Row 7
Adresses Record
Row 8
Shape ID:0
Transparent Rectangle
Starting x coordinate:1
Starting y coordinate:1
Ending x coordinate:770
Ending y coordinate:473
Width:2
Red Component:204
Green Component:204
Designer Tree
Grid Tree
Ready

Figure 16.10

Section 17

CONCLUSION

Problems during Implementation

Form Builder for Document Management System is a large scale project. During the project we faced some problems. Some of the problems were critical while some were moderate in nature. Some of the problems came up during the design stage while some revealed during the coding stage. The problems we faced during the implementation of this project are following:

1. Form Builder for Document Management System is a XML based form builder. XML is the essence of this project. Before starting the project, we were not aware of the XML and its sophisticated technologies so we had to learn XML and its technologies in detail before any activity concerning the project. There are not enough books available on this topic, neither in MCS library nor in the local market so we performed extensive browsing to collect the information about various topics of XML. Internet proved to be a very useful mean of learning XML. We downloaded so tutorials and research papers and carefully read them to grasp the concepts of XML and its technologies.
2. Another problem we faced in the project was the selection of appropriate technology for presentation of data in XML format. XML is content oriented language. Unlike HTML, which deals only with the presentation of data, XML takes care of semantics of the data as well. So in XML, a developer has to develop two kinds of structures. One kind of structures deal with the semantics of data while the other kinds of structures take care of presentation of the data. There are so many technologies available for presentation of data in XML. Some of them are cascading style sheets (CSS), XSLT, FOP and SVG. Selecting the right technology according to nature of our project was a difficult time at the time of project definition. Everyone in our syndicate was not appropriately aware of these sophisticated XML technologies. Each of them has its own merits and demerits. For example, CSS is very easy to use but it is difficult to implement in the code. Similarly XSLT is a very efficient technology but it’s not widely accepted and there are no XSLT editors available in the market. FOP is easy to program but it is a Java based technology developed by Apache.org. The last choice for us was SVG. When we examined SVG and gathered information about its various features we selected this format for the presentation of data in

our project. SVG is easy to program. It is widely accepted by a large number of vendors. A large number of SVG editors are available in the market and it is also supported by Internet Explorer 5.5 and above and Netscape Navigator 6 and above.

3. Before the start of the project, we were not proficient in Visual C++ which was supposed to be our main development tool. The project was assigned to us by Elixir technologies and we were asked to work with Visual C++ by the manager and our supervisor at Elixir. We were proficient in C++ and we had done a lot of object oriented programming in Java so we decided to accept the challenge. We consulted a large number of books and worked hard to learn enough VC++ to start the coding.
4. We have used Microsoft's Document/View architecture in the development of GUI. Before the start of the project, we were unaware of this very sophisticated and useful architecture so we had to learn the architecture. We consider the learning of this architecture as one of our major achievements from this project.
5. The main source of variable data, which act as an input to form builder, is a Flat file. Data, in a flat file, is presented in an unorganized manner. To facilitate the users, we decided to use a grid control for the presentation of data in an organized way. We modified a pre developed grid control. It was a very tough task. The grid control is indeed one of the most sophisticated grid controls present in the programming world. It contains a large number of classes. We understood the very lengthy code of the grid control and then we were able to modify the grid control according to our requirements. We consider the solution of this problem as a major milestone of our project because we really put in a lot of effort in solving this problem.

Achievements

Form Builder for Document Management System is a project with a lot of learning potential. During the project we achieved the following milestones and skills

- The concept of form builders
- Extensible Markup Language
- Visual C++ 6
- The concept of Flat File
- Addition of a grid control utility for flat file opening
- The concept of Scalable Vector Graphics (SVG)
- The concept of SVG viewers

After the completion of this project we are now able to take any project in the domain of variable print solutions. Also we have learned XML and SVG after which we are able to write XML pages for the use on internet. The project has really helped us in sharpening our existing programming skills and learning some modern and valuable concepts.

One of the biggest achievement of this project is that it has been tested and verified as standard and practical project by computer lab of PAKISTAN ATOMIC ENERGY COMMISSION (PAEC) and they have started using Form Builder in their labs

For preparing their different assignments .

FUTURE EXTENSIONS

Form Builder for Document Management was a very good experience. We really enjoyed our task because it was interesting. We planned and designed this project very carefully. We tried to cover each and every aspect but since we had to complete the project in a allotted time so we decided to leave some of the aspects. Leaving out some of the aspects does not make any effect on the core functionality of our project. Following extensions can be made in the Form Builder:

1. Animations can be incorporated in the form builder. A large number of hottest animation software ,like Flash 5.0 or 3d Studio Max, use vector graphics technology for the creation and management of animations. SVG supports the use of animation because it is a vector graphics based technology. Since Form Builder uses SVG for storing and retrieving the data so it is very easy to incorporate animations in the online forms.
2. Form Builder can be further extended to support various other XML based presentation technologies. For example, support for the XSLT can be incorporated or Cascading Style sheets can be supported by Form Builder.
3. Form Builder can be extended to support some very famous graphics format like jpeg, gif, targa, cdr, psd. Presently, Form Builder only supports bitmaps.
4. Form Builder can be used to support Adobe's famous Portable Document Format commonly known as PDF. A large amount of data on the web is available in the form of PDF supported pages so if Form Builder supports PDF then its interoperability and usefulness will be increased to a greater extent.
5. Form Builder can be developed as a Application Builder for the famous print streams like IBM's AFP or Xerox's VIPP. Incorporating the support for these famous print stream formats will certainly make the Form Builder a valuable variable printing tool.

Form Builder for Document Management System is really a good addition to the family of XML based software. XML is replacing HTML in many web applications so this project is an effort to learn and master various technologies of XML and many more additions can be made in this product to make it a commercial success.

Bibliography

- XML,How To Program , Dietel & Dietel
- XML Programming , Ivan Behrose
- XML and Java, Ivan Behrose
- Mastering Visual C++ 6.0, Micheal J Young
- Visual C++, Ivor Horton
- Teach Yourself Visual C++ in 21 Days, Chapman
- Inside Visual C++ 6.0
- C++, How to Program, Dietel & Dietel
- Programming in C++, Robert Lafore
- MSDN library
- http://www.adobe.com
- http://www.elixir.com
- http://www.w3c.org
- http://www.w3schools.com/xml.html
- http://www.w3schools.com/svg.html
- http://www.vbxml.com
- http://www.msdn.microsoft.com
- http://www.apache.org
- http://www.whatis.com

ACKNOWLEDGMENTS

We would like to thank Almighty Allah who gave us strength and wisdom to complete this project. We are indebted to Brig. Dr. Mohammad Akbar , who accepted himself as our project supervisor. He gave us some of his very precious time out of his very tight schedule and guided us. We are grateful to Mr Tauheed of Elixir Techonologies, who showed a great confidence in us while assigning this project. We are also grateful to Mr. Aswad Rehan, Mr. Babar Qaisarani, Mr. Aznan Hassan Khan, Mr Khurram Riaz and Mr. Adeel Abbas of Elixir Technologies for their guidance in the domain of XML and MFC programming. In the end, we would also like to thank our families and friends for their constant support and encouragement.

NC S.M.Khaliq-ur-Rahman Raazi

NC Wasim Imran

NC Junaid Arshad