Automated Tool for Software Maintenance

P-320

Project Report 1997 - 98

Submitted by

V. Pradeepa
K. Srilakshmi
G. Yuvaraj

Guided by

Mrs. S. Devaki, B.E.,

In partial fulfilment of the requirements
for the award of the Degree of
BACHELOR OF ENGINEERING
IN COMPUTER SCIENCE AND ENGINEERING
of Bharathiar University

Department of Computer Science and Engineering
Kumaraguru College of Technology
Coimbatore - 641 006

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore - 641 006

Project Work 1997 - 98

Name G. YUVARAJ, K. SRILAKSHMI, V. PRADEEPA Register No.

Certified that this is the Bonafide Record of the Project Work Done by

In partial fulfilment of the requirements for the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of the Bharathiar University.

[illegible]
Head of the Department

[illegible] 27/4
Guide

Submitted for the University Examination held on 27-04-98

[illegible]
Internal Examiner

[illegible] 27/04/98
External Examiner

DEDICATED TO
OUR BELOVED PARENTS

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We record our sincere thanks and profound gratitude to our beloved principal ***Dr.S.Subramanian,M.Sc.,(Engg.),Ph.D.,S.M.IEEE.,*** for having made available all the facilities to do this project.

We are elated to record our heartiest indebtedness to our beloved Professor and Head of the Department of Computer Science and Engineering, ***Prof.P.Shanmugam M.Sc.,(Engg.),M.S(HAWAII),S.M.IEEE,M.I.S.T.E.,*** for his support and encouragement during this project.

We are indeed previliged to have been under the light of ***Mrs.Devaki*** , ***B.E.*** We thank her for enlightening our thoughts and for her remarkable guidance.

We wish to thank our department staffs for their guidance .Our thanks to our department technicians for their extended co-operation.

We also thank ***Mr.Ramalingam, B.E.,*** DSQ for his help without whom the project would not have reached its stage.

We would like to extend our sincere thanks to our beloved parents for their encouragement and support all through the days of the project work.

We express our special thanks to our wonderful friends for their love and support.

SYNOPSIS

SYNOPSIS

Software maintenance, like software development, requires a combination of managerial control and technical expertise. Since software maintenance is a very important phase of software development, the activities that a software maintainer may become involved in span the entire spectrum of software engineering. Automated tools to support the software maintenance include technical support tool and managerial support tool.

Automated tools to support the technical aspects of software maintenance span the spectrum from analysis and design tools to implementation tools. Version Control Systems, which is one of the automated tools for software maintenance, provides basics to our project.

Version Control System is used to track the history of each module in a system by recording which modules and which version of which modules comprise which system releases. This tool can also be used to associate source modules with object modules, to record which versions use which modules, and to provide system protection by preventing updates to master files that have not been approved by the change control board. Also, a Version Control System can coordinate experimental versions of a system when several maintenance programmers are simultaneously modifying the same module or sub system.

CONTENTS

CONTENTS

		Page No.
1.0	**INTRODUCTION**	1
1.1	Automated tools for software maintenance.	1
1.2	Version Control System – An Automated tool.	2
2.0	**SYSTEM CONFIGURATION**	3
3.0	**PROJECT DESCRIPTION**	4
3.1	What is Version Control System.	4
3.2	Operations of Version Control System.	5
3.2.1	Working Directory	5
3.2.2	Check In	5
3.2.3	Check Out	5
3.2.4	View file	5
3.2.5	Get file	5
3.2.6	Difference	6
3.3	Version Control System Administration	6
3.3.1	User List maintenance	6
3.3.2	Project List maintenance	6
3.4	How to work with VCS?	6
3.5	Services offered by VCS	7
4.0	**IMPLEMENTATION**	9
4.1	VC++ - The powerful tool for Windows application	9

4.2 Technical notes 10

4.2.1 Document – View Architecture 11

4.2.2 Single Document Interface(SDI) 12

4.2.3 Multiple Document Interface(MDI) 13

4.2.4 ODBC 14

4.2.5 OLE 15

4.3 OLE Automation Server/Client 17

4.3.1 Server Application 20

4.3.2 Client Application 21

4.4 Project Design 22

4.5 Output Screens

5.0 CONCLUSION 24

6.0 FUTURE EXPANSION 25

7.0 BIBILOGRAPHY 26

8.0 PARTIAL SOURCE CODE IN VC++ 27

INTRODUCTION

INTRODUCTION

1.1 Automated Tool for Software Maintenance

Since software maintenance is a microcosm of software development, the tools, techniques and activities of software maintenance span the entire product life cycle. Difficulties encountered in software maintenance and the social stigma attached to software maintenance are largely due to lack of systematic planning for software maintenance during the development process, the failure of programmers and mangers to properly organize for software maintenance activities for an failure to provide the necessary tools and techniques for software maintenance.

Planning for maintenance, developing the software product to enhance maintainability, proper organization of maintenance programmers and maintenance activities, and provision for maintenance tools can result in vast improvements in software quality, programmer productivity, and programmer morale.

Automated tools for software maintenance include both technical and managerial tools. Tools to support the technical aspects of software maintenance span the spectrum from analysis and design tools to implementation tools to debugging and testing tools. Automated tools include

- Text editors
- Debugging aids
- Cross_ reference generator
- Linkage editors
- Comparators
- Complexity metric calculators
- Version control systems
- Configuration management databases

Version Control System, which is one of the most necessary tools for the software maintenance, has captured our interest. This also seems to very apt. We have developed our project based on the concepts of Version Control System.

1.2 About Version Control System

In software development environment it is essential to maintain the project files securely, preserving the changes made by each and every user to the files. The problem arises when two or more users try to update the same file simultaneously. Version Control System provides solution to this problem. Version Control System supports team application development, enabling team members to share files, modify them independently and later merge the changes.

Version Control System only stores the changes instead of the whole file, for every change made by the developer. This results in saving disk space for medium and large file in most cases. It can prevent accidental replacement of information or files themselves. Files can be shared across projects, platforms. Portability issues can be tracked. By tracking which program use which modules, code reusablity is possible.

Version Control System keeps track of versions, changes, date and time of modification, acting as a historian. Version Control System also assists some of the highlighting features such as proper maintenance, co-ordination and timely completion of the project in an efficient manner.

Version Control System administrator takes care of security measures to the project files by providing access rights to the users. Maintenance of the user list and project list is also an essential function of the administrator to avoid unauthorised access to the files.

Version Control System is flexible enough to support any size of project and any number of users.

SYSTEM CONFIGURATION

SYSTEM CONFIGURATION

Hardware

- A pentium processor with 120MHZ ,16MB RAM .
- A hard disk with 1GB space.
- A analog RGB monitor, which can support VGA resolution.
- Windows 95/Windows NT operating system.

Software

- Visual C++ 5.0
- MS Access 97.

PROJECT DESCRIPTION

PROJECT DESCRIPTION

One of the automated tools for software maintenance, the Version Control System answers the following questions,

How many versions of each file exist?

How do the versions differ?

What is the history of each component of each version of each product?

Is a given old version still used by some user?

3.1 What is Version Control System?

Version Control System, which is an automated tool for software maintenance, is basically a project-oriented system. A project is collection of interrelated files that are stored in the system. The user can add, delete, edit and share files through projects. A project has much in common with the operating system directories, but there are significant differences. A project can have subprojects. The project tree hierarchy has C:\SERVER\PROJECT\ ... as its root project. Project paths are separated by slashes (\), for example

C:\SERVER\PROJECT\newproj......

Users cannot work in the files stored in the Version Control System.The file must be moved out of the system to work with. The user cannot work with the master copy. The Version Control System provides each user with a copy of the file to read or change. Each time someone checks in changes to a file, the system not only stores the changes, but the history of changes as well.

3.2 Operations of Version Control System:

3.2.1. *Working Directory* :

The user cannot actually work on a file within the Version Control System. When user wants to work on a file, he must obtain a copy of the file from the VCS and have place to put it in. That place is the user's WORKING DIRECTORY. The working directory is his personal base for a project. He can make changes only if the copy of the file available in working directory.

3.2.2. *Check In Operation:*

The file that is stored in version control system database and not available for modification is called the CHECKED IN file. Usually When the modification are made to the local copy of the file available on the Working Directory, we store this file back to the Version Control System. This process is called CHECK IN operation. This operation stores the changes as a new version of file.

3.2.3. *Check Out Operation:*

The file that is reserved for work by user, is called the CHECKED OUT file. Usually this is the local copy of the file in working directory. The user can make changes to this file. By default only one user at a·time can check out a file.

3.2.4. *View File Operation:*

View File operation opens the latest version of the file for the user. Here user can only view the file but cannot edit it. The file is opened in the read only mode with the help of Notepad.

3.2.5 *Get File Operation:*

In Get File operation also the latest version of the file is opened, but in the edit mode. The user can make changes only to the checked out file.

3.2.6 Difference Operation:

In the Difference operation we compare the latest version of the file with the selected version of the file. This comparison shows all the changes that are made in the latest version.

3.3. Version Control System Administration:

The Version Control System administrator is the person who has responsibilities to maintain user list, security options, database and project list.

3.3.1. User List Maintenance:

Maintenance of the user list is the foremost duty of the administrator. Options are available to add new users to the user community. The user's name and access rights are also specified during add operation. Assignment of project is also done while adding the user. Deletion of user is also possible. The user's password and the project assigned to him can be altered when required.

3.3.2. Project List Maintenance:

The administrator has to add projects and subprojects in Version Control System. Files must be added to the project in order to perform Check In and Check Out operations. At the same time deletion of the unwanted files is also possible. The file can be viewed by using Notepad. The file history that contains the number of versions created to the file, the date and time of the version created, file status and the user who created the version can also be displayed during the addition of the files to the project.

3.4. How to work with VCS?

The steps to be followed when working with VCS are mentioned below.

- Create one or more projects in VCS in which the files are to be organised.
- Set a working directory for each project.
- Add files form operating system directory into VCS projects.

- Check the files Out of VCS into Working Directory.
- Use an editor to make changes to the file in the Working Directory.
- Check In the files back into the VCS from the Working Directory.

3.5. Services Offered By VCS:

- Library Service
- History Service
- Security Service

3.5.1. Library Service:

This system prevents accidental deletion of information or files themselves. The files are organised into hierarchy of projects and subprojects. Checking Out files are made easy and comments can be added to describe the changes. It enables two or more users to share files across projects or platforms. When multiple projects use a file multiple copies of the files need not be used. VCS sees to that only one master copy is stored and that it is the most recent version. Cross platform applications can be developed and portability issues can be tracked. VCS helps in modular or object oriented code development by keeping track of which programs use which module.

3.5.2. History Service:

Protecting changes to each file is important. So it is necessary to keep a record of changes made, who made them and where they were made. VCS keeps a detailed history of each file and project in addition to any comment s that you enter.

As a historian, VCS performs functions like tracking versions and changes users make to files, tracking date and time of changes to all files in the database and displaying differences between two versions of a file.

3.5.3. ***Security Service:***

Controlling access to the files is a critical component of maintaining source control. By default VCS maintains security on projects so that each new user has read-write access or read-only access. There is an administrator for this system who enables security. Access rights can be set on user and project. Whenever a user executes a command, the system checks to determine his rights. The VCS also provides additional security for the VCS administrator, the administration program.

IMPLEMENTATION

IMPLEMENTATION

4.1 VC++ - The Powerful Tool for Windows Applications

Version Control System is a Windows application implemented using Microsoft Visual C++ 5.0 using Windows 95 operating system. Visual C++ provides a host of features that makes developing Windows applications easier. The MFC library is part of Visual C++ provides lot of classes commonly used. The Version Control System uses the Client/Server model. Some of the components of VC++ are described below

- ***Microsoft Developer Studio 97:***

Developer Studio is a Windows-hosted Integrated Development Environment (IDE) that's shared by Visual C++, Microsoft Visual J++, Microsoft Visual Basic and several other products. Docking windows and configurable toolbars, plus a customizable editor that runs macros are one part of Developer Studio. The online help system (InfoViewer) works like a Web browser.

- ***AppWizard:***

AppWizard is a code generator that creates a working skeleton of a Windows application with features, class names, and source code file names that you specify through dialog boxes. AppWizard code is minimalist code; the functionality is inside the application framework base classes.

- ***ClassWizard:***

ClassWizard is a program that's accessible from Developer Studio's View menu. ClassWizard takes the drudgery out of maintaining VC++ class code. Need a new class, a new virtual function, or a new message-handler function? ClassWizard writes the prototypes, the function bodies, and the code to link the Windows message to the function. ClassWizard can update that you write, so you avoid maintenance problems.

♦ ***Microsoft Foundation Class Library (MFC):***

The Microsoft Foundation Class Library (MFC) is an "application framework" for programming in Microsoft Windows. Written in C++, MFC provides much of the code necessary for managing windows, menus, and dialog boxes; performing basic input/output; storing collections of data objects; and so on. All you need to do is add your application-specific code into this framework. And, given the nature of C++ class programming, it's easy to extend or override the basic functionality the MFC framework supplies.

Main features of MFC are,

- ♦ A C++ interface to the Windows API
- ♦ General-purpose classes (non-Windows-specific) class, including
  - # Collection classes for lists, arrays, and maps
  - # A useful and efficient string class
  - # Time, time span, and date classes
  - # File accesses classes for operating system independence
  - # Support for systematic object storage and retrieval to and from disk
- ♦ A "common root object" class hierarcy
- ♦ Streamlined Multiple Document Interface(MDI) application support
- ♦ Support to OLE (Object Linking and Embedding)

4.2 Technical Notes

VCS project is dissolved into two applications, SERVER application and CLIENT application.All the administrative operations are implemented in server application and the remaining operation are carried out by client application. The communication between client and server is established through OLE automation.

Key concepts such as Document –View architecture, SDI, MDI, ODBC, OLE automation server/client, which are used for the development of our project are explained in detail in forthcoming sections.

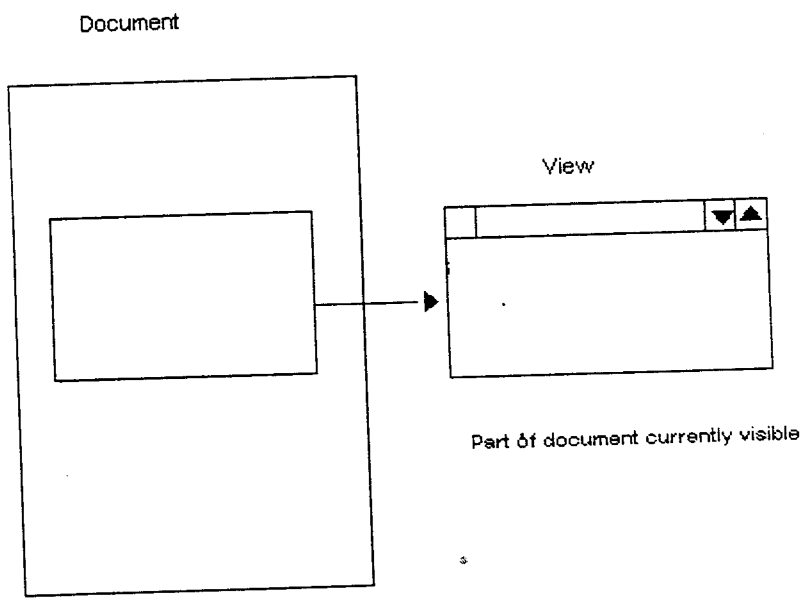
4.2.1 ***Document-View Architecture*:**

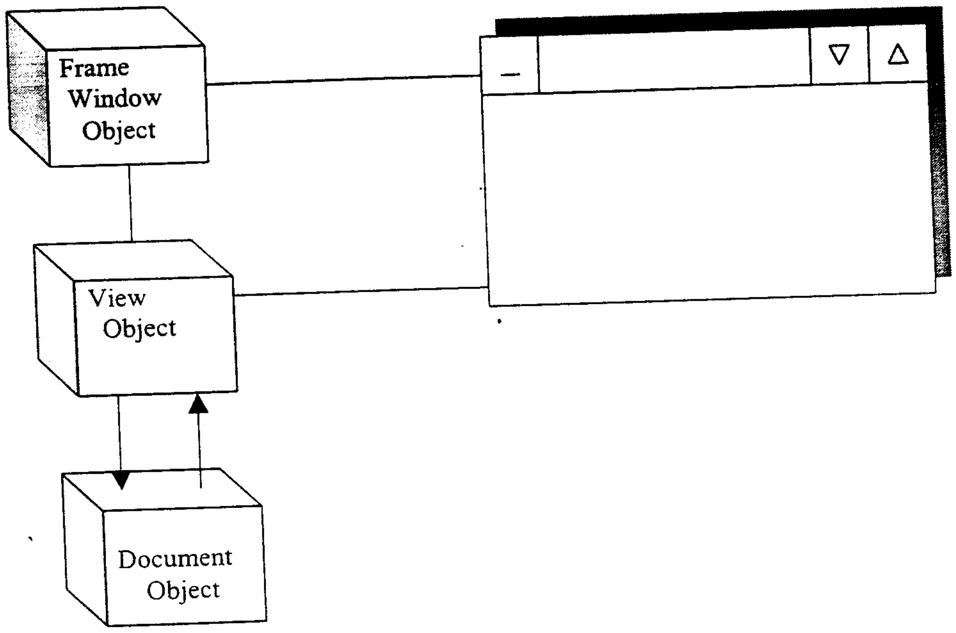
The document/view implementation in the class library separates the data itself from its display and from user operations on the data. All changes to the data are managed through the document class. The view calls this interface to access and update the data.

Documents, their associated views, and the frame windows that frame the views are created by a document template. The document template is responsible for creating and managing all documents of one document type.

The key advantage to using the MFC document/view architecture is that the architecture supports multiple views of the same document particularly well.

The following figures show the relationship between the document and its view and frame and view.

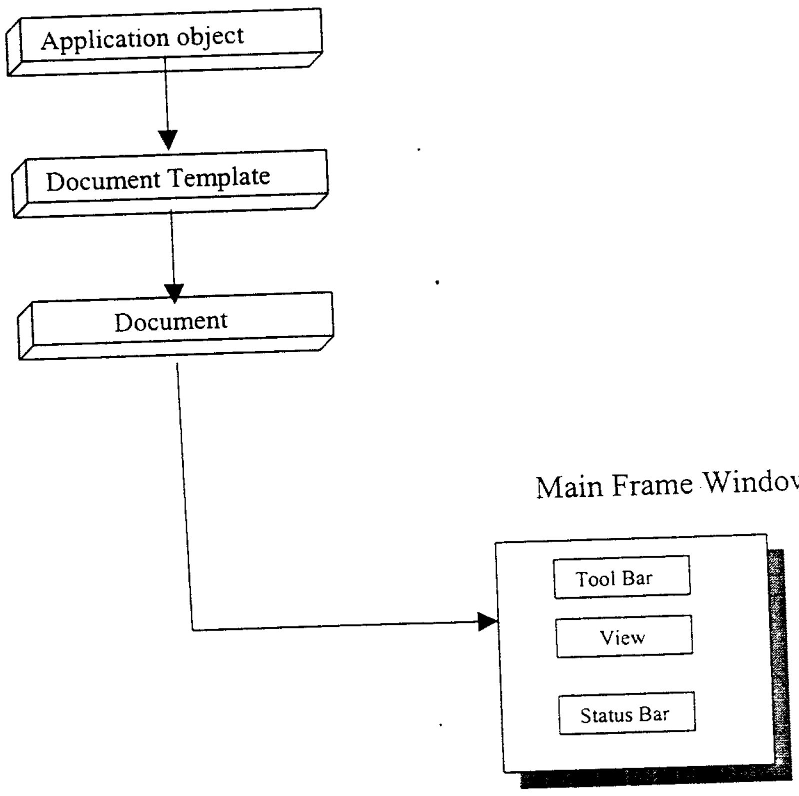




4.2.2 *Single Document Interface (SDI) :*

The SDI has only one document and can have multiple views. The document class in the case of the SDI is the CDocument class and it has one or more view classes each ultimately derived from CView class. A complex handshaking process takes place among the document, the view, and the rest of the application framework. The client program of this project is implemented as SDI application.

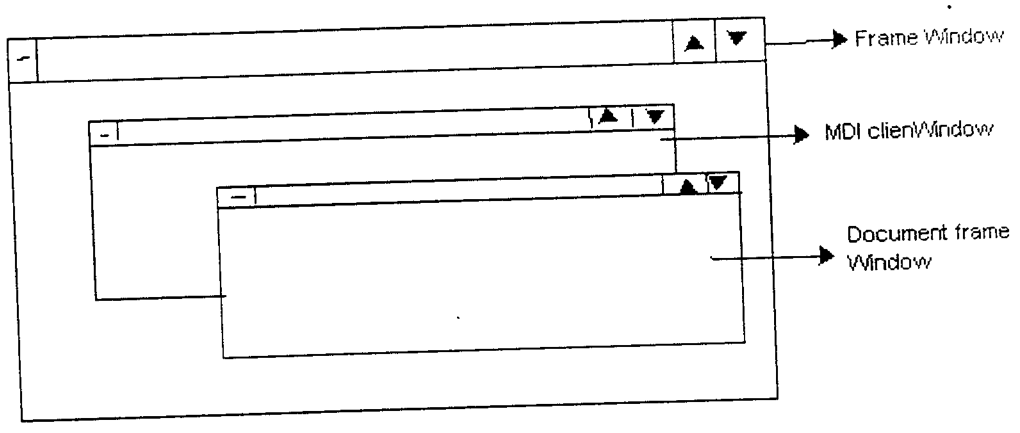
Objects in a SDI application



4.2.3. ***Multiple Document Interface (MDI) :***

MDI applications allow multiple document frame windows to be open in the same instance of an application. An MDI application has a window within which multiple MDI child windows, which are frame windows themselves, can be opened, each containing a separate document. In some applications, the child windows can be of different types, such as chart windows and spreadsheet windows. In that case, the menu bar can change as MDI child windows of different types are activated.

MDI Frame Windows and children



4.2.4. ***Open DataBase Connectivity (ODBC):***

The Microsoft Open Database Connectivity offers programming languages like C, C++ to access database systems like MS-Access, MS-Foxpro, Borland's Paradox, Powersoft's Powerbuilder etc. The ODBC contains a powerful set of DLLs that offer standard database applications an interface. They act as an interface between the porgram and the database system. The server program in our project to store and access the user and file information maintained by the MS-Access database efficiently uses ODBC.

4.2.5. ***Object Linking and Embedding (OLE):***

OLE is a mechanism that allows users to create and edit documents containing items or "objects" created by multiple applications.

OLE documents, historically called compound documents, seamlessly integrate various types of data, or components. Sound clips, spreadsheets, and bitmaps are typical examples of components found in OLE documents. Supporting OLE in any application allows users to use OLE documents without worrying about switching between the different applications; OLE does the switching for you.

OLE incorporates many different concepts that all work toward the goal of seamless interaction between applications. These areas include the following:

Linking and Embedding Linking and Embedding are the two methods for storing items created inside an OLE document that were created in another application. For general information on the differences between the two, see the article OLE Background: Linking and embedding.

In-Place Activation (Visual Editing) activating an embedded item in the context of the container document is called in-place activation or visual editing. The container applications interface changes to incorporate the features of the component application that created the embedded item. Linked items are never activated in-place because the actual data for the item is contained in a separate file, out of the context of the application containing the link.

Automation Automation allows one application to drive another application. The driving application is known as an automation client or automation controller, and the application being driven is known as an automation server or automation component.

Compound Files Compound files provide a standard file format that simplifies structured storing of compound documents for OLE applications. Within a compound file, storages have many features of directories and streams have many features of files. This technology is also called structured storage.

Uniform Data Transfer Uniform Data Transfer (UDT) is a set of interfaces that allow data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data. UDT forms the basis for data transfers by drag and drop. UDT now serves as the basis for existing Windows data transfer, such as the Clipboard and dynamic data exchange (DDE).

Drag and Drop Drag and drop is an easy-to-use, direct-manipulation technique to transfer data between applications, between windows within an application, or even within a single window in an application. The data to be transferred is simply selected and dragged to the desired destination. Drag and drop is based on uniform data transfer.

Component Object Model The Component Object Model (COM) provides the infrastructure used when OLE objects communicate with each other. The MFC OLE classes simplify COM for the programmer. COM is part of ActiveX, as COM objects underlie both OLE and ActiveX.

The VCS project is implemented using OLE automation concept even though OLE incorporates several concepts, which are mentioned above.

Having seen the concepts used in this project, Version Control System, the MS-Access database, the server and the client program are dealt with in detail in the following passages.

MS-Access Database:

The server process to hold user and project information uses the MS-Access database. The server uses a database file by the name csdata.mdb. The tables and their attributes are explained below.

- Table USER

Attribute	**Type**
User_id	Varchar(15)
User_Name	Varchar(15)
Password	Varchar(15)
ProjName	Varchar(15)

The User_id field helps to identify each user in the user list. The user-type field specifies 'A' for administrator and 'C' for clients. The user_Name fields contain the user name. The Password field is used for security reasons. Every time the user wants to login the password entered by the user is checked against this Password field. The proj-Name field gives the name of the project that the user is assigned to.

- Table PROJECT

Attribute	**Type**
Projname	Varchar(50)
CreationDate	Date/Time
TotalSubDir	Int
TotalFiles	Int

4.3 .OLE Automation Server/Client

4.3.1 Automation Clients:

Automation makes it possible for your application to manipulate objects implemented in another application, or to expose objects so they can be manipulated. An automation client is an application that can manipulate exposed objects belonging to another application. The application that exposes the objects is called the Automation server. The client manipulates the server application's objects by accessing those objects' properties and functions.

The class COleDispatchDriver provides the principal support for the client side of Automation. Using ClassWizard, you create a class derived from COleDispatchDriver.

You then specify the type-library file describing the properties and functions of the server application's object. ClassWizard reads this file and creates the **COleDispatchDriver**-derived class, with member functions that your application can call to access the server application's objects in C++ in a type-safe manner. Additional functionality inherited from **COleDispatchDriver** simplifies the process of calling the proper Automation server.

4.3.2. Automation Server:

An Automation server is an application that exposes programmable objects to other applications, which are called Automation clients. Exposing programmable objects enables clients to automate certain procedures by directly accessing the objects and functionality the server makes available.

Exposing objects this way is beneficial when applications provide functionality that is useful for other applications. For example, a word processor might expose its spell-checking functionality so that other programs can use it. Exposure of objects thus enables vendors to improve their applications' functionality by using the ready-made functionality of other applications.

By exposing application functionality through a common, well-defined interface, Automation makes it possible to build applications in a single general programming language like Microsoft Visual Basic instead of in diverse, application-specific macro languages.

Support for Automation Servers

ClassWizard, AppWizard, and the framework all provide extensive support for Automation servers. They handle much of the overhead involved in making an Automation server, so you can focus your efforts on the functionality of your application.

The framework's principal mechanism for supporting Automation is the dispatch map, a set of macros that expands into the declarations and calls needed to expose methods and properties for OLE. A typical dispatch map looks like this:

```
BEGIN_DISPATCH_MAP(CMyServerDoc, COleServerDoc)
  //{{AFX_DISPATCH_MAP(CMyServerDoc)
  DISP_PROPERTY(CMyServerDoc, "Msg", m_strMsg, VT_BSTR)
  DISP_FUNCTION(CMyServerDoc, "SetDirty", SetDirty, VT_EMPTY, VTS_I4)
  //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

ClassWizard assists in maintaining dispatch maps. When you add a new method or property to a class, ClassWizard adds a corresponding DISP_FUNCTION or DISP_PROPERTY macro with parameters indicating the class name, external and internal names of the method or property, and data types.

ClassWizard also simplifies the declaration of Automation classes and the management of their properties and operations. When you use ClassWizard to add a class to your project, you specify its base class. If the base class allows Automation, ClassWizard displays controls you use to specify whether the new class should support Automation, whether it is "OLE Createable" (that is, whether objects of the class can be created on a request from an OLE client), and the external name for the OLE client to use.

ClassWizard then creates a class declaration, including the appropriate macros for the OLE features you have specified. ClassWizard also adds the skeleton code for implementation of your class's member functions.

AppWizard simplifies the steps involved in getting your automation server application off the ground. If you select Automation support in AppWizard's OLE Options page, AppWizard adds to your application's InitInstance function the calls required to register your Automation objects and run your application as an Automation server.

SERVER APPLICATION:

The server program is a MDI application .It is implemented as a full-fledged OLE server. A single copy of the server has to be running before the clients get tô run. Apart from the Administration activities it takes care of User Setup and project Setup.The server program recevies requests from the OLE clients and processes their requests. It also maintains an MS-Access database as a backend to store the user and project infomation. The server uses ODBC connectivity to access the database from the VC++ program. The server offers a neat to the administrator.

The interface is achieved by making the server a simple MDI application. Two views are provided in the server window. Splitter window is used to create a neat interface with multiple views. The two views provided are Tree_view (derived from CTreeView MFC class) and List-view (derived from CListview class) help in simulating a file explorer .The treeview is used to display the projects in the Version Control System. It also displays the hierarchy of projects and subprojects. The list-view is used to display the list of files in the project cuurently expanded in the tree-view. The details of the porjects are accessed from MS-Access database via ODBC. The user list can be viewed when needed.

Since our server program is made full-fledged OLE server and client program OLE enabledclient, CLIENT\SERVER model is enforced by default. OLE provides uniform data transfer between clients and server. The communication required between the server and clients such as the server receiving requests from the clients, server sending back results to the clients is taken care of the OLE server and OLE client generated by VC++.

The server maintains an MS-Access database and accesses it using ODBC.When user list needs to be displayed, the USER table in the database is accessed and the result of the query is displayed. The administrator using the server process may need to update The user list, either by adding users and modifying the user details. For these activities the table USER is updated. The server program uses user setup porcess to associate the project files with the user.

CLIENT APPLICATION:

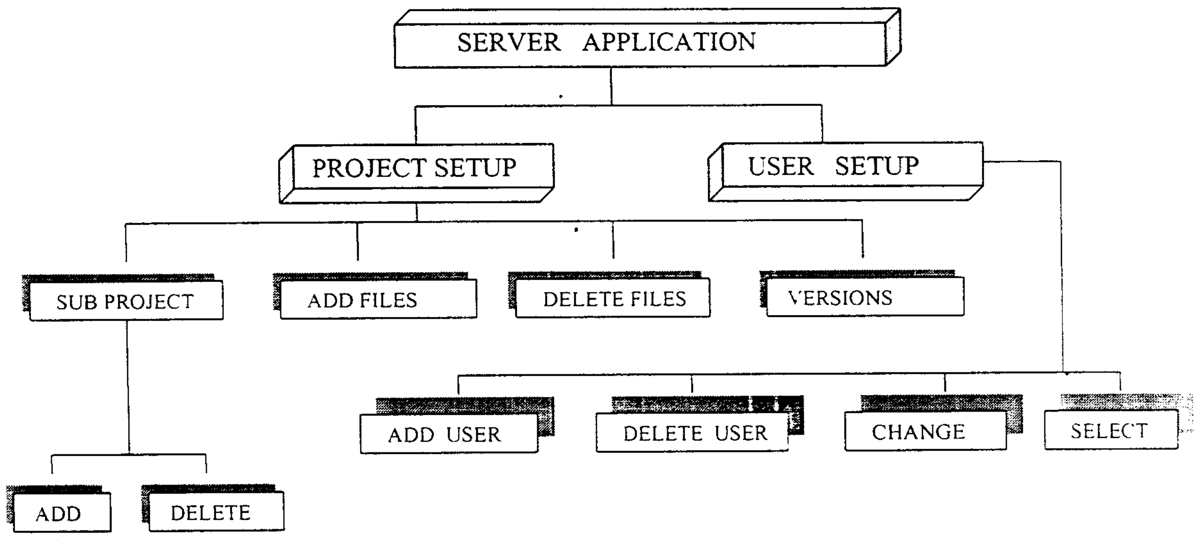
The client program is implemented as a SDI application. This program uses OLE to embed client information from server. Many instances of client program can run at the same time. The clients send requests to the server program and receive the result. It provides a neat interface to display different versions of a file. The cilent program also integrates a good text editor that the user can use to change the checked out files in his working directory.

The user has various options provided by the client. The client program interface uses Splitter window to show multiple views. He may want to give a file to the Version Control System, check out file, check in file etc. The client is required to pass to the server some data regarding the user's request. Since the client is an OLE enabled client it embeds the data into the OLE server and thus the client and server communicate.

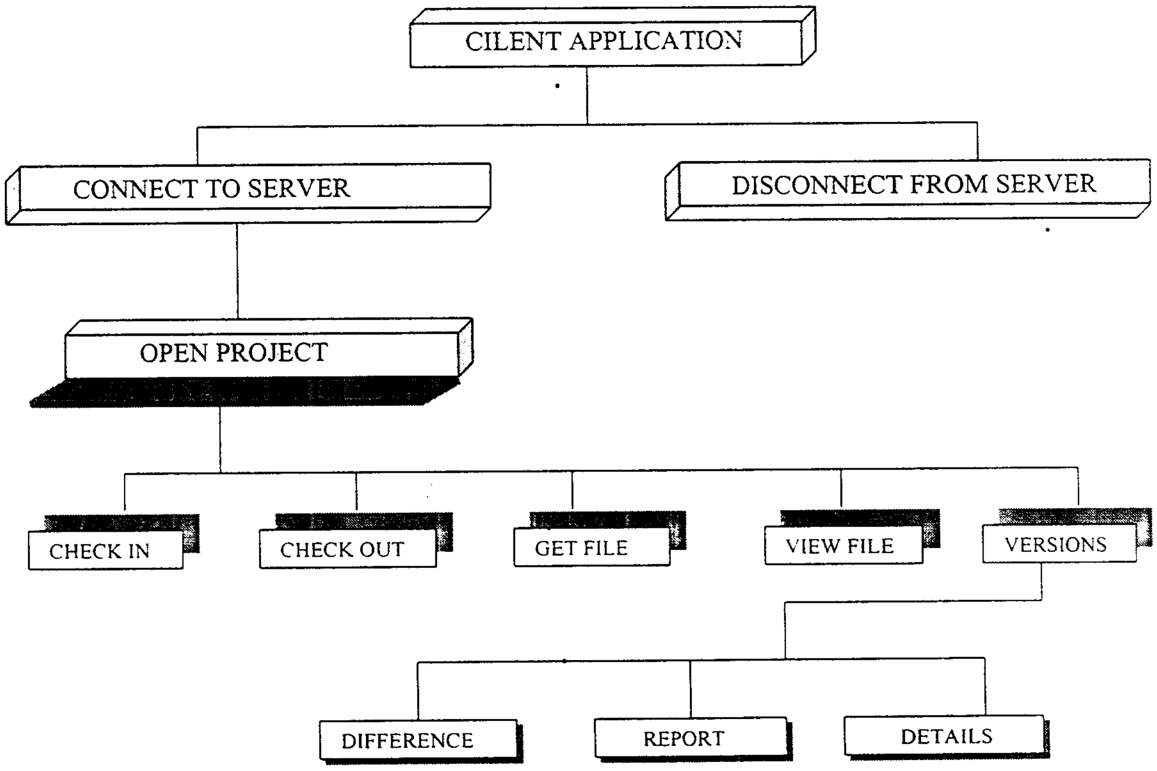
PROJECT DESIGN

4.4 Project Design

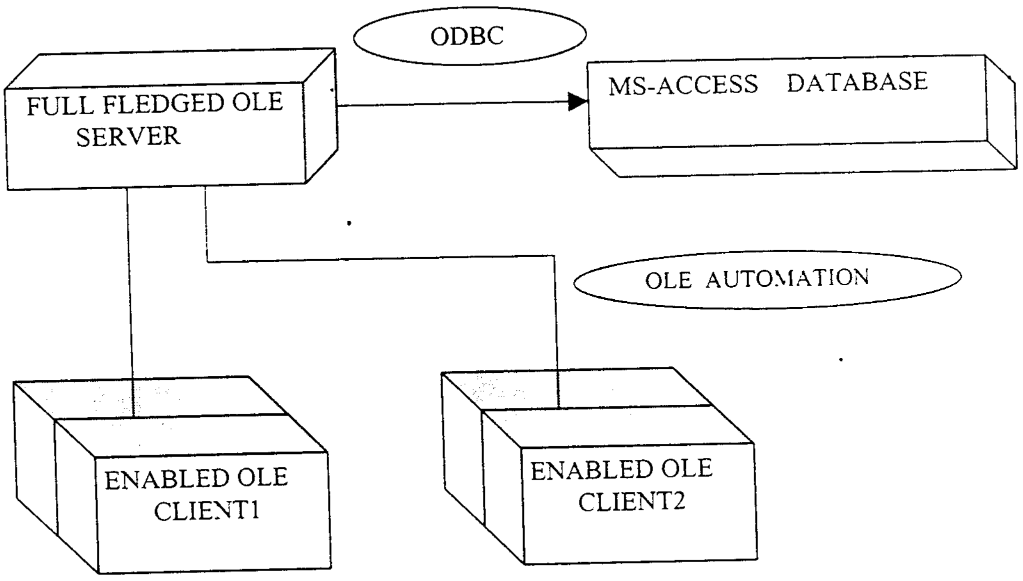
Server Design:



Client Design:



INTERFACE DESIGN:



OUTPUT SCREENS

SERVER SCREEN

CLIENT SCREEN

CONCLUSION

CONCLUSION

A breakthrough in the software industry is necessary to cope with the exploding IT Industry. There is absolute necessity for a proper co-ordination between the developers of the software. There is also necessity to assure the speedy completion of the project and satisfactory customer services. Our project succeeds in satisfying this requirement. Implementing the concepts like file sharing, code reusability and proper security measures are some of the objectives that are aimed at and they are achieved.

We also came to a conclusion that the windows graphic user interface was excellent and is suitable for our application. Windows also facilitated us to make the software easily understandable by any lame user, no prior knowledge of computing is necessary.

FUTURE EXPANSION

FUTURE SCOPE

"There are miles to go before we sleep ". We have just made a start. It is the job of the future engineers to enhance this project to attain its final colors.

The software attains its final stage only after the integration of the different modules of the developers. Integration requires **merging** of different version of a file. Merging is not a easy task. When two or more different changes are made to the same line of the file a **conflict** occurs. Merging becomes tedious during conflict. Finding out a better way to resolve the conflict is left to the future development.

Multiple CheckOut is another criteria in Version Control System that is simultaneous checkouts of same file by two or more users. Multiple check out is not possible unless it is enabled by the Version Control System Adminstrator. By implementing this idea the project may reach its pinnacle. It is the future developers who have to facilitate multiple check out.

BIBLIOGRAPHY

BIBILIOGRAPHY

1. Inside Visual C++ (Fourth edition) – David J.Kruglinski
2. Microsoft Visual C++ 5 (power tool kit) – Richard C.Leinecker
3. Teach yourself Visual C++ in 21 days – Ori Gurewich and Nathan Gurewich
4. Visual Source Safe Manual

PARTIAL SOURCE CODE IN VC++

```
// CSServer.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "CSServer.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "IpFrame.h"
#include "CSServerDoc.h"
#include "CSServerView.h"
#include "OpenProjDialog.h"
#include "direct.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCSServerApp

BEGIN_MESSAGE_MAP(CCSServerApp, CWinApp)
	//{{AFX_MSG_MAP(CCSServerApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
	ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	//ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCSServerApp construction

CCSServerApp::CCSServerApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
	m_TotalUsers = 0;  // RAMU
	m_UserSet = new CUserSet(&m_CSDataBase);
	m_ProjectSet = new CProjectSet(&m_CSDataBase);
	m_FilesSet = new CFilesSet(&m_CSDataBase);

	m_TotalProj = 0;

	pathDirectory = "C:\\CSServer\\Project\\";
```

```
        sTempDirPath = "C:\\CSServer\\Temp\\";

        bOpenFile = FALSE;

}

CCSServerApp::~CCSServerApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
        if(m_UserSet != NULL)
                delete m_UserSet;
        if(m_ProjectSet != NULL)
                delete m_ProjectSet;
        if(m_FilesSet != NULL)
                delete m_FilesSet;
        m_CSDataBase.Close();
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCSServerApp object

CCSServerApp theApp;

// This identifier was generated to be statistically unique for your app.
// You may change it if you prefer to choose a specific identifier.

// {3768300B-ACD9-11D0-8D9D-0000C0283725}
static const CLSID clsid =
{ 0x3768300b, 0xacd9, 0x11d0, { 0x8d, 0x9d, 0x0, 0x0, 0xc0, 0x28, 0x37, 0x25 } };

/////////////////////////////////////////////////////////////////////////////
// CCSServerApp initialization

BOOL CCSServerApp::InitInstance()
{
        // Initialize OLE libraries
        if (!AfxOleInit())
        {
                AfxMessageBox(IDP_OLE_INIT_FAILED);
                return FALSE;
        }

        AfxEnableControlContainer();

        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.
```

```
#ifdef _AFXDLL
        Enable3dControls();                    // Call this when using MFC
                                                      in a shared DLL

#else
        Enable3dControlsStatic();    // Call this when linking to MFC
                                                  statically

#endif
        LoadStdProfileSettings();  // Load standard INI file options
                                            (including MRU)

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CMultiDocTemplate* pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(
                IDR_CSSERVTYPE,
                RUNTIME_CLASS(CCSServerDoc),
                RUNTIME_CLASS(CChildFrame), // custom MDI child frame
                RUNTIME_CLASS(CCSServerView));
        pDocTemplate->SetServerInfo(
                IDR_CSSERVTYPE_SRVR_EMB, IDR_CSSERVTYPE_SRVR_IP,
                RUNTIME_CLASS(CInPlaceFrame));
        AddDocTemplate(pDocTemplate);

        // Connect the COleTemplateServer to the document template.
        //  The COleTemplateServer creates new documents on behalf
        //  of requesting OLE containers by using information
        //  specified in the document template.
        m_server.ConnectTemplate(clsid, pDocTemplate, FALSE);

        // Register all OLE server factories as running.  This enables the
        //  OLE libraries to create objects from other applications.
        COleTemplateServer::RegisterAll();
                // Note: MDI applications register all server objects
                                            without regard

                //  to the /Embedding or /Automation on the command line.

        // create main MDI Frame window
        CMainFrame* pMainFrame = new CMainFrame;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
                return FALSE;
        m_pMainWnd = pMainFrame;

        // Parse command line for standard shell commands, DDE, file open
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);

        // Check to see if launched as OLE server
```

```
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
        // Application was run with /Embedding or /Automation.
                                    Don't show the
        //  main window in this case.
        return TRUE;
}

// When a server application is launched stand-alone, it
                                    is a good idea

//  to update the system registry in case it has been damaged.
m_server.UpdateRegistry(OAT_INPLACE_SERVER);
COleObjectFactory::UpdateRegistryAll();

// The following line will stop the application from creating a
// new document at startup

cmdInfo.m_nShellCommand = CCommandLineInfo::FileNothing;

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
        return FALSE;

m_nCmdShow = SW_SHOWMAXIMIZED;

// The main window has been initialized, so show and update it.
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();

// RAMU
if (m_CSDataBase.IsOpen())
        m_CSDataBase.Close();

CString strConnect("ODBC;");

strConnect += "UID=";
strConnect += "";
strConnect += ";";

strConnect += "PWD=";
strConnect += "";
strConnect += ";";

try {
        m_CSDataBase.Open(_T("Student Registration"), FALSE, FALSE,
                strConnect);

}
catch (CDBException* pEx)
{
        CString strMessage(_T("Could not open database: "));
```

```
                strMessage += "CSDATA";
                AfxMessageBox(strMessage);
                pEx->Delete();
                return FALSE;
        }

        return TRUE;
}
////////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CCSServerApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}
/////////////////////////////////////////////////////////////////////////////
// CCSServerApp commands

void CCSServerApp::OnFileOpen()
{
        COpenProjDialog projectDlg;
        if(projectDlg.DoModal() == IDOK)
        {
                //CWinApp::OnFileOpen();
                //CString tempCString;
                //tempCString = pathDirectory;
                //tempCString += theApp.m_OpenedProj[theApp.m_TotalProj - 1];

                if(!TestWhetherProjDirExists()) // If the project directories
                                                // doesn't exist, return
                {
                        AfxMessageBox("Project directory doesn't exists");
                        return;
                }

                theApp.bOpenFile = TRUE;
                OpenDocumentFile("readme.txt");
        }
}

// Test whether the project directory exists
BOOL CCSServerApp::TestWhetherProjDirExists()
{
        BOOL return_value;
        TCHAR szcurr_dir[256];
        CString proj_path = theApp.pathDirectory;
        proj_path += theApp.m_OpenedProj[theApp.m_TotalProj - 1];

        // Save the current working directory
        ::GetCurrentDirectory(sizeof(szcurr_dir),szcurr_dir);
        if(_chdir(proj_path) == -1)
                return_value = FALSE;
        else
                return_value = TRUE;
        // Restore the current working directory
        ::SetCurrentDirectory(szcurr_dir);
        return return_value;
}
```

```
// CSServer.h : main header file for the CSSERVER application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

// RAMU
#include "UserSet.h"
#include "ProjectSet.h"
#include "FilesSet.h"

/////////////////////////////////////////////////////////////////////////////
// CCSServerApp:
// See CSServer.cpp for the implementation of this class
//

class CCSServerApp : public CWinApp
{
public:
	CCSServerApp();
	~CCSServerApp();

	// RAMU
	int m_TotalUsers; // This keeps track of number of clients connected
	                                          // to the server


	CDatabase m_CSDataBase;
	CUserSet  *m_UserSet;
	CProjectSet		*m_ProjectSet;
	CFilesSet *m_FilesSet;

	CString	m_OpenedProj[25]; // Maximum 25 projects can be opened
	int	m_TotalProj;
	BOOL bOpenFile;

	CString pathDirectory;
	CString sSelectedNodePath;
	CString sTempDirPath;

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCSServerApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation
```

```
        COleTemplateServer m_server;
                // Server object for document creation

        //{{AFX_MSG(CCSServerApp)
        afx_msg void OnAppAbout();
        afx_msg void OnFileOpen();
        //}}AFX_MSG

        BOOL TestWhetherProjDirExists();

        DECLARE_MESSAGE_MAP()
};

extern CCSServerApp theApp; // RAMU

/////////////////////////////////////////////////////////////////////////////
```

```
// CSServer.odl : type library source for CSServer.exe

// This file will be processed by the Make Type Library (mktyplib) tool to
// produce the type library (CSServer.tlb).

[ uuid(3768300C-ACD9-11D0-8D9D-0000C0283725), version(1.0) ]
library CSServer
{
        importlib("stdole32.tlb");

        //  Primary dispatch interface for CCSServerDoc

        [ uuid(3768300D-ACD9-11D0-8D9D-0000C0283725) ]
        dispinterface ICSServ
        {
                properties:
                        // NOTE - ClassWizard will maintain property
                                                      information here.
                        //    Use extreme caution when editing this section.
                        //{{AFX_ODL_PROP(CCSServerDoc)
                        //}}AFX_ODL_PROP

                methods:
                        // NOTE - ClassWizard will maintain method
                                                      information here.
                        //    Use extreme caution when editing this section.
                        //{{AFX_ODL_METHOD(CCSServerDoc)
                        [id(1)] boolean ValidateUser(BSTR* UserInfo);
                        [id(2)] void DisconnectUser();
                        [id(3)] boolean CheckInFile(BSTR* FileInfo);
                        [id(4)] boolean CheckOutFile(BSTR* FileInfo);
                        [id(5)] boolean GetFile(BSTR* FileInfo);
                        [id(6)] boolean GetVersion(BSTR* VerInfo);
                        [id(7)] boolean GetFileInfo(BSTR* FileInfo);
                        [id(8)] boolean GetReserveFn1(BSTR* GeneralInfo);
                        [id(9)] boolean GetReserveFn2(BSTR* GeneralInfo);
                        [id(10)] boolean GetReserveFn3(BSTR* GeneralInfo);
                        //}}AFX_ODL_METHOD
        };
        //  Class information for CCSServerDoc

        [ uuid(3768300B-ACD9-11D0-8D9D-0000C0283725) ]
        coclass Document
        {
                [default] dispinterface ICSServ;
        };

        //{{AFX_APPEND_ODL}}
};
```

```
// DiffDlg.cpp : implementation file
//

#include "stdafx.h"
#include "CSClient.h"
#include "DiffDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDiffDlg dialog


CDiffDlg::CDiffDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDiffDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDiffDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CDiffDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDiffDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDiffDlg, CDialog)
        //{{AFX_MSG_MAP(CDiffDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDiffDlg message handlers

BOOL CDiffDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        char *file=new char[100];
        char *t=new char[100];

        char a1[80][2],a2[80][2];
```

```
 CStringArray f1,f2;
int z=0;
      strcpy(t,theApp.nVersions);
while(t[z]!='\0')
      {
          file[z]=t[z+3];
                              z++;
                              }

       CString file1,file2;
       file1=theApp.sProjectPath;
       file1+=theApp.sProjectName;
       file1+="\\";
       file2=file1;
       file1+=theApp.nVersions;
       file2+=file;
       delete [] file;
       delete [] t;

  CStdioFile fp1(file1,CFile::modeReadWrite);
        CStdioFile fp2(file2,CFile::modeReadWrite);

  f1.SetSize(80);
        f2.SetSize(80);
  int   i=0;
        while(fp1.ReadString(f1[i])!=NULL)
        {
                  ++i;

        }

        AfxMessageBox(f1[i]);



        int k=0;
        while(fp2.ReadString(f2[k])!=NULL)
        {
        ++k;
        }

        int m=0,pos=0,c,p1=0,m1=0;


while(m<i && m1<k)
{

        c=strcmpi(f1[m],f2[m1]);

          if(c==0)
     {                 strcpy(a1[m],"n");
```

```
                    strcpy(a2[m1],"n");

            }
 if(c!=0)
 {
            pos=m1;
            ++pos;
     while(c!=0 && pos<=k)
            {
                         c=strcmpi(f1[m],f2[pos]);

          if(c==0 && strcmpi(a2[m1-1],"n")==0)
                {
                               for(int s=m1;s<pos;++s)
                                   {
                                    //strcpy(a1[s],"i");
                                    strcpy(a2[s],"i");
                                         //a1[s+1]='\0';
                                         a2[s][1]='\0';


                                   }
                                   break;

                }
           else if(c==0)
                 {
                            for(int s=m1;s<pos;++s)
                                  {
                //strcpy(a1[s],"c");
                                   strcpy(a2[s],"c");
                                   a1[s][1]='\0';
                                         a2[s][1]='\0';
                                  }
                                  break;
           }
                ++pos;
             }//while

             if(pos<=k && m<i)
             {
               ++m;
         ++pos;
         m1=pos;
             }
 //          else
    //   {
             //         pos=0;
             //}

    }//if
```

```
if(c!=0 && pos>k)
{
                    p1=m;
                    ++p1;
                     while(c!=0 && p1<=i)
                            {


                  c=strcmpi(f1[p1],f2[m1]);


           if(c==0 && strcmpi(a2[m1-1],"n")==0)
                                {
                           for(int s=m;s<p1;++s)
                                         {
                                    strcpy(a1[s],"d");
                                                 a1[s][1]='\0';
                                      //a2[s+1]='\0';
                                     //strcpy(a2[s],"i");
                                         }
                            break;
                                }
          else if(c==0)
                               {
                            for(int s=m;s<p1;++s)
                                          {
                                                     strcpy(a1[s],"c");
                                      // strcpy(a2[s],"c");
                                                     a1[s][1]='\0';
                                       a2[s][1]='\0';
                                          }
                                          break;
                               }
                               ++p1;
                             }//while

    if(p1<=i && m1<k)
    {
                               ++m1;
                   ++p1;
                   m=p1;
                   }
   // else
   //{
                   //          p1=0;
      //}
          }//if


          if(p1>i && pos>k)
```

```
            {

        strcpy(a1[m],"c");
                      strcpy(a2[m1],"c");
                      ++m;
                      ++m1;


            }
      else if((pos==0 && p1==0) || c==0)
            {
               ++m;
                      ++m1;

            }
}

if(i>k)
{
        for(int h=m;h<=i;++h)
   {
          strcpy(a1[h],"d");
           a1[h][1]='\0';
        }

}

if(i<k)
{
        for(int h=m1;h<=k;++h)
   {
          strcpy(a1[h],"i");
           a1[h][1]='\0';
        }

}

AfxMessageBox("Listcontrol");
int LineIndex=0;
CListCtrl *List=(CListCtrl*)GetDlgItem(IDC_LISTDIFF);
for(int q=0;q<=i;++q)
{

        if(strcmpi(a1[q],"d")==0)
        {
                List->InsertItem(LineIndex,"deleted");

                //cout<<"d  -"<<f1[q]<<endl;
        }
           else if(strcmpi(a1[q],"c")==0)
```

```
                List->InsertItem(LineIndex,"changed");
                    //cout<<"c  -"<<f1[q]<<endl;

        else
            List->InsertItem(LineIndex,f1[q]);
                    //cout<<f1[q]<<endl;

              LineIndex++;
    }

    /*for(int q1=0;q1<=k;++q1)
    {

              if(strcmpi(a2[q1], "i")==0)
                     cout<<"i  -"<<f2[q1]<<endl;

                 else if(strcmpi(a2[q1],"c")==0)
                     cout<<"c  -"<<f2[q1]<<endl;


        else
                     cout<<f2[q1]<<endl;

     }*/

              // TODO: Add extra initialization here

              return TRUE;  // return TRUE unless you set the focus to a control
                            // EXCEPTION: OCX Property Pages should return FALSE
     }
```

```
protected:

        // Generated message map functions
        //{{AFX_MSG(CDiffDlg)
```