THEATRE VISION

A Computerized Solution for Light Show Based On DMX512 Protocol

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

By

SURENDRA KUMAR GAUTAM

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)

MAY, 2002

CNS Comsoft Systems Pvt. Ltd.

25, Sant Nagar, East of Kailash, Top Floor,
New Delhi-110065. INDIA
Tel. No. : 91-11-6237218,
Fax No. : 91-11-4679807
E-mail : naveenc@giasdl01.vsnl.net.in

Ref. No. CNS/MKT/05..252 Date : 07th May 2002

CERTIFICATE

This is certify that Mr. **Surendra Kumar Gautam s/o Shri Ram Hans** student of M.C.A. curriculum of **Indian Institute of Technology Roorkee(IITR), Roorkee** , has undergone practical training in our organization from Jan 2002 to May 2002 for the partial fulfillment of the award of Master of Computer Application, **Indian Institute of Technology Roorkee(IITR), Roorkee**.

During the above said period he undertook the project titled **" THEATRE VISION"** Light and sound Control System as a team member and worked on **"Design Phase and DMX 512 Protocol"**.and performed very well. He has successfully completed it under my personal guidance.

This work has not been submitted to any other Institution / University for award of any degree / diploma.

[illegible]

NAVEEN CHAUHAN
(DIRECTOR)
CNS COM SOFT SYSTEM PVT. LTD.

CANDIDATE'S DECLARATION

I hereby declare that the project work entitled **"THEATRE VISION A Computerized Solution for Light show Based on DMX512 Protocol "** carried out during MCA dissertation in partial fulfillment of their requirement for the award of the degree of Master of Computer Application (MCA-III) and submitted in the Department of Mathematics, Indian Institute Of Technology, Roorkee, is an authentic record of my own work carried out under the guidance of **Dr. VINOD KUMAR Professor, Department of Electrical Engineering**, Indian Institute Of Technology Roorkee,Roorkee.

The matter embodied in this project has not been submitted by me elsewhere for the award of any other degree.

Dated: May 2002 **(Mr. SURENDRA KUMAR GAUTAM)**

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

[illegible]

(Mr. Naveen Chauhan)
CNS COMSOFT SYSTEMS Pvt.Ltd.
SANT NAGAR, NEW DELHI
Dated: May 2002

[illegible]

(Dr. VINOD KUMAR)
Professor,
Department Of Electrical Engineering
I I T, Roorkee
Dated: May 2002

ACKNOWLEDGEMENT

I take this opportunity to thank prof. H.G.SHARMA, head of the Department of MATHEMATICS, IIT Roorkee, Roorkee, Dist Haridwar, and all other staff members for their invaluable guidance through out The MCA curriculum.

I am thankful to **Professor Vinod Kumar**, (senior prof. In IIT Roorkee) for his supervision and guidance throughout the project training.

I am also thankful to **MR. NAVEEN CHAUHAN** Director of CNS Comsoft System Ltd. Who provide me a wonderful environment of software development and for availing me all the available resources for the project Development .

I am thankful to **Professor R.C. Mittal** (prof. In IIT Roorkee) for guidance throughout the project training.

I am also thankful to **MR. SUSHIL KUMAR DOHARE** (Ph.D. scholar).

He work as advisor throughout the project training and provide guideline In project development.

Surendra Gautam

Surendra Kumar Gautam

CONTENTS

ABSTRACT

Our world is full of light and sound. Even if we are not able to perceive the full range of acoustic waves, the human ear is nevertheless a fantastic organ and we can differentiate between precisely 1378 different tones. Similarly the power of vision, the human eye is undoubtedly the greatest of nature's gift to mankind. And since the beginning of electronic sound and light reinforcement with introduction of computers, the past 50 years have seen a major change in worldwide market of broad casting systems and innovative lighting systems.

The Light industry with the concept of a Lightshow has come along way and now with introduction of computers in this arena, more realistic, fast and innovative shows are being designed. THEATRE VISION is a computerized solution and new introduction to a Light control show.

This report gives a deep insight into the system design and DMX512 Protocol, which is used in the development of the software, which handles this system. Through this we can handle different 512 dimmers for changing there intensity means fade out and fade in the dimmers. And can provide many relay effects like Smoke Generator, light projector, video projector, audio system, cloud generator, This fully compliant with USITT standard for DMX 512 data transmission. It is user friendly and designed for inexperienced computer operators. One can exploit full potential of DMX 512 protocol using this software. All 512 channels can be engaged at one go. One can design/save/edit/auto run a show live and fully automatically.

One can also know about the Console to operate channels manually. Up to 500 scenes can be stored in particular show. Any channel can be programmed as Chaser or Flasher. In case of power failure or during rehearsals, show can be resumed from any desired scene.

For the development of this project we have used the Microsoft Visual C++ 6.0 and used MFC and SDK. The main thing is concept of "Threads". This is simulation of the real system to the software module. We can pre-design the dimmer's show and we can test it at system and can remove the bugs in the show at the design level. This is off site designing of show and can be

changed the intensity of light just by sliding the vertical bars, which is provided in the window of show designed.

For storing the light data we have generated the .CUE , . SEQ and LRN files which is specially designed for compressed data format.

INTRODUCTION
ABOUT THE COMPANY

CNS COMSOFT SYSTEMS PVT. LTD. is a broad based professional setup with a team of dedicated professionals consisting of Chartered Accountants, Cost Accountants, MCAs, System Engineers and qualified personnel with extensive experience in the field of Turnkey Project implementation involving wide ranging hardware and software and communication protocols involving broad based communication equipment to provide customized solutions for various OLTP operations using worldwide communication networks.

Another division is involved in trading of agriculture pulp, flowery culture, export of fish lings. Also we undertake prototype development of polymer shoe lasts.

CNS is the fastest growing completely integrated industrial house involved in multiple businesses. The latest edition to the CNS family is fully computerized unit managing data bases for body shopping across the globe, with special emphasis on man power resources from south east Asia, India Bangladesh and lately a few CIS countries.

Backed by widespread experience and expertise, it possesses the added advantage of being able to offer COMPLETE SOLUTIONS - right from planning and automation of an organization, installation of office equipment, development/installation of software up to the training of personnel.

The list of Corporate Business Houses which have been served by CNS include:

*** PHOENIX OVERSEAS PVT. LTD.**
*** LIFE INSURANCE CORPORATION OF INDIA**
*** HAL**
*** BHARAT HEAVY ELECTRONICS LTD.**
*** GAS AUTHOURITY OF INDIA LIMITED**
*** MINISTRY OF COMMERCE**
*** GANDHI PEACE FOUNDATION**
*** DUNCUNS (G.P.G. ENTERPRISES)**
*** BILT (TOSCANA LASTS LTD.)**
*** VINTAGE GROUP**
*** MINISTRY OF BROADCASTING**
*** SIEL**

The software development undertaken by CNS is using ORACLE, SYBASE, Windows based programming tools, OOPs, Assembly language, Power builder, Watcom SQL, Visual C++, DB2, Networking and Multimedia applications.

The working platforms for software development include HP-900, PC 486 / PENTIUM-III 1000 MHz, VAX-11, DEC-Alpha. The Operating System includes MS-DOS, VMS, UNIX, ULTRIX, Windows 98 and Windows NT.

Main objectives of the company

To carry on the business of designing, developing, buying, selling, exporting, importing system solution and software use in various applications in India and abroad.

To install, maintain, procure rights, buy, sell, let on hire, import, export or otherwise deal in computer hardware and accessories, communications, Process control and office automation equipment.

To carry on business of system integration, buying, selling, exporting, importing, assembling, leasing equipment and system in the telecom and electronic sector.

To carry on business of advisors and consultants to individuals, firms, companies, corporate bodies, societies, organizations, undertakings, institutions, matters, relating to communications, computers, networks, Managerial and Information Systems (MIS).

To enter into various strategic alliances and collaborations with various individuals, firms, companies, government, local authorities and others in India or abroad for development of new avenues in the software or other allied lines.

To establish and operate data and information processing centers and bureaus and to render services to customers in India and elsewhere by processing their jobs as data processing centers giving out computer machine time on hire or license basis.

To enter into contracts with individuals, companies, partnership firms, association of persons, societies, local authorities, bodies for providing Internet services, E-mail facilities and faxing facilities through computer on License basis in India or abroad.

CHAPTER: 1

INTRODUCTION TO PROJECT

Our world of light and sound...

A Lightshow is basically an engineer's media and not a director's media. The concept is a powerful visualization, theme display on very large stage (or multiple number of small stages) of size even exceeding 2000 feet. The colorful display of Light supported by effects like smoke generator, cloud generators, lighting generators, audio effects, video projectors, slide display, etc. Is a powerful approach of explaining a theme, a subject, a story or whatsoever the message is? The show designer has to design the light cues synchronized with sound so that with a change in audio sound by a visible change in lights to focus on to some particular object, slide, video clip or person can be visualized.

In India, song and drama division of ministry of information and broadcasting is organizing Lightshows with a large network of about 27 branches in the country since it's first establishment in 1952 and the first Lightshow in 1967.

Son-et-lumeir the Lightshow without characters changed into theatre Panasonic, the Lightshow with characters in 1969 when song and drama division evolved this new concept. With the latest Light equipment, the division today organizes a large number of shows throughout the country.

In India, since the first Lightshow organized in 1965 at red fort directed by Chetan Anand, the Light industry in India has come long way from the analog light controls to the dimmers, then the cue processors for lights, and finally the intelligent dimmers which are the latest innovation and sound has gone digital, from single track to 24 track recording, studios with the latest sound equipment's, mixers, MIDI interface, computerized sound controls and what not.

Light industry with the concept of Lightshow has come a long way and has still many horizons to reach.

1.1 PROBLEM DEFINITION

Why this project............?

Since a Lightshow can not exceed a few lights and effects if manually controlled, computerization is demanded. So for analog cue processors widely used in this area propose limitations like only a few dimmer access and a few effects besides their other drawbacks. Further longer shows of just duration of 2-3 hours take the cue designer days tom design the light cues synchronized with sound. This also restricts show organizers to go for multiple shows in a month. Since the analog system imposes no-edit at last moment, if a flaw is remaining in the cue design, it cannot be corrected at the last stage.

Hence a computerized solution for pre-designing a show off site with synchronized sound decks and dimmers, which has a cue processor, a dimmer controller interface, which can record these shows onto a safe backup media for a perfect performance and finally which can allow edit of a pre-designed Lightshow is demanded. And this defines the THEATRE VISION.

DEVELOPING SYSTEM

- **EXISTING SYSTEM**
- **PROPOSED SYSTEM**
- **PROPOSED SYSTEM REQUIREMENTS**

CHAPTER: 2

DEVELOPING SYSTEM

2.1 EXISTING SYSTEM

A Lightshow is normally executed by analog cue processors, which have to be manually done for fading in lights for show. This requires the user to prepare all cue sheets on paper and finally he has to practices these cues before going on for show. This restricts the show designer to handle more dimmers, lights, controls at the same time longer shows are critical to design the existing system has all these drawbacks besides time consumption for designing a show which is days and sometimes weeks for the show designer.

A somewhat digital solution as given by martin which works only for martin lights but fails when a general concept of Lightshow comes up. It does not include many effects and restricts to 24 dimmers, which is also the manual limit of the show designer. Beside the complexity of show design, the dimmer controller does not support DMX-512 protocol. Being DOS based software with 16-bit application it is comparatively very slow.

So practically no system exist in accordance tom the user requirements. A general light and show designing software which can store shows as files on hard disk and works on 512 dimmers, with 1000 scenes and 1,00,000 cues the THEATRE VISION is an innovation in the arena of Light industry.

2.2 PROPOSED SYSTEM

Project outline......

THEATRE VISION is a complete solution to pre-design a Lightshow off-site. It can then finally run a Lightshow controlling the intelligent dimmers synchronized with sound and can handle relay effects for smoke generator, video projector, cloud generator, lightning generator, slide projector, audio effects, etc.

The software has an inbuilt interface using DMX-512, the lighting protocol. DMX-512 is a light control system for applications like rock shows, stage shows, and fashion shows, Lightshows, display, etc. It incorporates three components i.e. cue processor, data link and dimmers.

The system proposes a complete hardware and the interfacing software capable of handling 512 dimmers and 8 sound decks portable enough for performance on-site and using compressed data storage by having 26 hours of a show requiring at the maximum of 8.5mb of disk capacity. The installation software will require about 30mb of free disk space on the hard disk.

The hardware team had designed the complete hardware after a complete R&D of five years. The remaining part of the system was software interfacing the designed hardware, a show designing module, scenes, storing the pre-design show as files (.CUE, .SEQ and .LRN file formats) and finally the show execution of synchronized light, sound, relay controls etc.

It is important to note that dimmer controller has a data receiving speed of 4800 bit/sec, audio varies from 5000 bit/sec to 200k bit/sec, video beyond 100k bit/sec and the software working on a Pentiums processor based PC should synchronized all these to result in a live Lightshow. The software will be a 32-bit application compatible with Windows 98/ Windows NT.

2.2.1 Proposed software features...

- *Testing at each scene levels*
- *Cue designing and testing at design level*
- *A maximum of 1,00,000 cues giving a show design possibility of shows for up to 88 hours and beyond*
- *Selected from a maximum of 1000 preset design scene. From these any 25 scenes can be fade-in, fade-out or locked simultaneously*
- *8 effects can be simultaneously used in a cue with the preset scene combinations (relay effects can be cloud generator, smoke generator, video projector, slide projector, audio effects, lightning generator, flashers, etc.)*
- *Specially designed compressed file formats. .CUE, .SEQ, .LRN for storing light sequences for scene and effects for final show execution*
- *compressed file storing requires just 8.5 Mb to run a light sequence for 26 hours*
- *maximum 512 dimmer control*
- *serial output with RS 485 signals*
- *specially designed for DMX-512*
- *Windows 98 and MFC based user friendly 32 bit multimedia application compatible with Windows NT*

2.2.2 Proposed system output....

- ***8 bit asynchronous data***
- ***4800 bit/sec***
- ***RS 485 compatible signal levels for standard interface***
- ***XLR sockets(3-pin) HCG***
- ***SVGA output for monitor***

2.3 REQUIREMENT ANALYSIS

Since the system was unique of its kind and there is no such existing system in the market so a trial survey was done with a few Light pioneers of the industry. It was concluded that the show designers badly needed a computerized solution to design cues. A very user friendly GUI based system which could interface the DMX controller, send data signals to relays so as to finally design a perfect show.

All these requirements led to a choice of visual C++ ver 6.0 – 32 bit programming tool for Window 95/ Windows NT/ Macintosh. Besides being an integrated tool supporting GUI, it also extends support to Microsoft foundation classes, WIN 32 SDK, WIN 32 API and inline programming of other languages, further the tool supports multimedia, ODBC and OLE programming too.

The system requirement for development of project was a 12-FlyPack dimmers rack with 4 delays, Serial Port connecting interface, DMX-512 Protocol standards, a Multimedia Pentium processor based PC with 2 GB HDD, CD-ROM drive, etc.

Time to time consultation with people from the industry was also necessary.

The major Studios and Light 'n' Sound Pioneers who were a part of survey, requirements and consultation included A V Workshop, Thukral Stagecrafts, Modern Radios, etc.

Besides studying the existing system, a computerized solution developed by MARTIN the #1 company dealing in lighting products was also studied. All its shortcomings like it

could not work beyond 24 dimmers, system requirements were unique and not general were all understood well before the software design phase begins. Since audio required a lot of disk space and similar was for light sequence, requirements of compressed file format for storing light Cues was also justified. Since design a show through these existing systems was so typical, that each Lightshow if designed was by a single person, then that show could never run without the presence and actual handling by the same person. So a general software requisite, which could on its own run aLightshow without user's intervention once designed, was defined.

SYSTEM ANALYSIS AND DESIGN

- ***System design***
- ***Class description***
- ***Technical issue***
- ***Testing***

CHAPTER:3

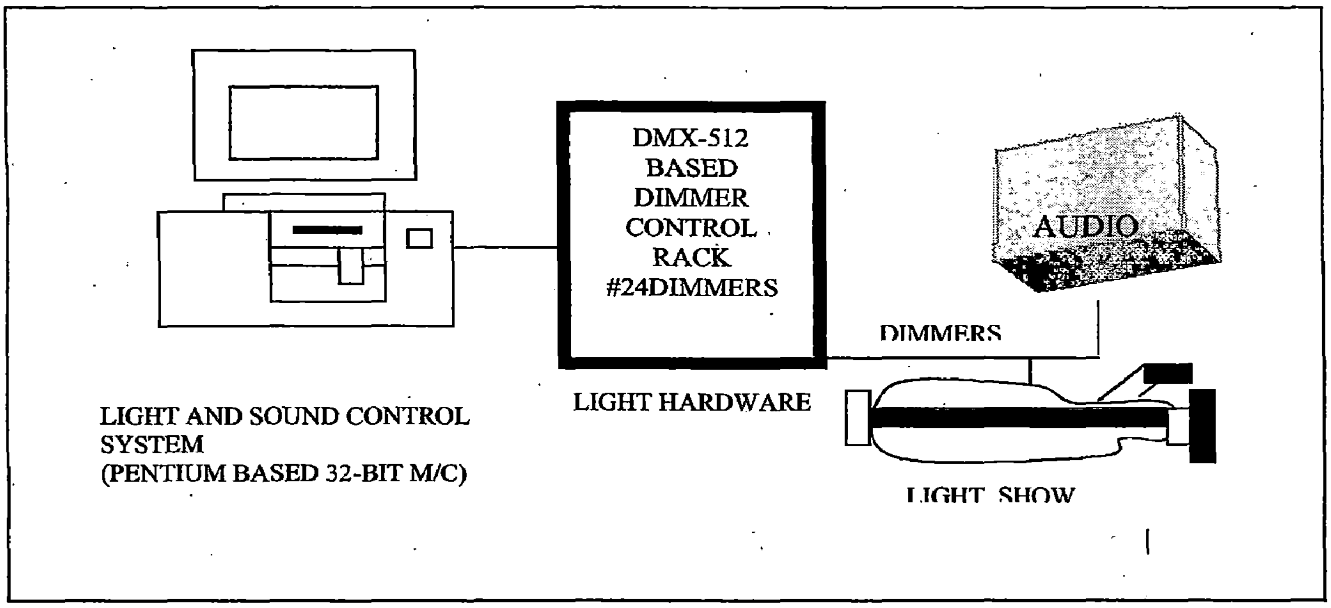
SYSTEM ANALYSIS AND DESIGN

3.1 SYSTEM ANALYSIS

The THEATRE VISION can be sub classified into three main areas:

- Show designer's Imaginations (to design light and sound show) and hence the input light and sound cue data.
- Software interface (cue processor) to actually handle this crude data, store into compressed data files (.CUE, .SEQ and .LRN files formats), process the data to help user test cues, sequences, etc. For a final run show
- The out being a perfectly synchronized show execution from the software interface as data signals to the intelligent dimmer controllers supporting the DMX-512 protocol for light data signals.

The following figure shows the main actual flow of the system:


DMX-512 BASED DIMMER CONTROL RACK #24DIMMERS
AUDIO
DIMMERS
LIGHT AND SOUND CONTROL SYSTEM (PENTIUM BASED 32-BIT M/C)
LIGHT HARDWARE
LIGHT SHOW

The three basic components of the system are:

- CUE PROCESSOR
- DATA LINK
- DIMMERS

The other device which are the secondary components of the system include the audio systems, all supporting audio device, remote effect like smoke machine, cloud projector, flashers, video projectors, etc

3.1.1 CUE PROCESSOR

The cue processor is an intelligent system capable of designing, processing and transmitting the various light values for different dimmers the system is also capable of storing the information indefinitely in the memory for further use and modification in the form of cue files. It is also capable of switching on remote effects like smoke machine, cloud projector, flashers, video projectors, etc. The individual compressed binary files formats used for the storing the input cue information Viz. . CUE and .SEQ are explained in detail in the later part of the report.

3.1.2 DATA LINK

The normally used data link for DMX-512 system is RS 485 interface in asynchronous mode. The link is a two-wire cable of low impedance. An optional data link for DMX-512 can be fiber optic cable. Through it is important to notice that data signals through the serial port output of the PC are feeble and hence amplification is required by connecting it with 9V DC source (SMPS of the PC avail this voltage) and hence becomes distinguishable.

3.1.3 DIMMERS

Dimmers unit is an intelligent microprocessor based controller which interprets DMX-512 signals from cue processor and arranges the firing angle of triacs / SCR's accordingly so as to output the required value of light on the lamps.

In general the cue light processor control offers a very high speed real time processing to give 4800 bit/sec baud rate for dimmer control on RS-485 signal levels over an XLR female with which up to eight racks of 12 dimmers each (i.e. 8*12=96 dimmers) can be stacked.

Another excellent feature of the system is that its signals can be recorded with audio on a tape using VISTASONIC TM interface available separately in the market. This feature is very useful for prerecorded light and sound show execution without actually using the computer on site. So broadly the output is as follow:

- ***Channels for dimmers***
- ***Analog to digital converter (by PC)***
- ***Scenes at maximum***
- ***Cues at maximum***
- ***Maximum files storing requirement 10 MB each show.***
- ***Serial asynchronous data transmission***
- ***VISTASONIC TM compatible***
- ***Flypack TM compatible***

3.2 SYSTEM ANALYSIS AND DESIGN

A set of components that interact to accomplish a specific task called system. The major components of our THEATRE VISION s have already been discussed. The design phase for the system refers to the process of planning a new system or one to replace or complement of an existing system. But before this planning is done the existing system needs to be understood. System analysis, then is the process of gathering information or recommend improvements to the system.

Since THEATRE VISION has an analog system as the existing system, the analysis, which was required, was to actually understand the hardware, its functioning, the DMX-512 protocol and the concept of a light and sound show.

3.3 SYSTEM DEVELOPMENT LIFE CYCLE

The systems development life cycle is basically all the phases, which have undergone throughout the development and implementation of the complete system as whole.

Excluding the hardware development portion, the four basic activities involved in the life cycle of the software development of this project are:

- ***Requirement analysis***
- ***Design***
- ***Coding***
- ***Testing***

Requirements analysis is done in order to understand the problem which the problem which the software systems to solve. The developer has to satisfy all the user requirements and identifying what is needed from the system. It should be noted that it does not include how the system will achieve its goals.

The process of gathering the information about the client's need is called fact-finding techniques. The fact-finding techniques may include interviews, questionnaires, observations, etc.

Requirement analysis has individually been discussed in earlier part of this report. In case of THEATRE VISION the fact-finding techniques were interviews with some pioneers of light and sound industry. By review of another customized software from MARTINE for its own lights and by observing a few light and sound shows actually happening in the capital.

Design...!

The design phase is required to plan the problem, as specified by the requirement analysis done by the developer. This is the first step in moving from problem domain to the solution domain. So the design phase has two areas of study:

- ***System design***
- ***Detailed design***

System design is responsible for...

- Identifying the modules
- Identifying the interaction of modules with each other
- Data source
- Flow of data from source to destination
- All file formats involved for software development
- Output file formats (.CUE, .SEQ and .LRN)

3.4 SYSTEM DESIGN

Since the problem as gathered by the requirement analysis phase can not be handled as a whole hence the basic principle of problem partitioning is used.

The THEATRE VISION (Light and Sound control system) design hence required problem partitioning into four areas:

- New show design for scene settings
- Edit / open an existing show for scene setting
- process designed scene setting for cue design
- Final show execution

Besides this the file formats which store light data i.e. scenes and cues, must also be designed. This format as documented earlier in this report have been specially designed for this project and do not follow any other file format structure available in the industry.

. CUE and .SEQ file formats used for storing scene setting and cues respectively and .LRN created and used only at runtime of show. having compressed binary file format structure and these are individually described in the later part of this report.

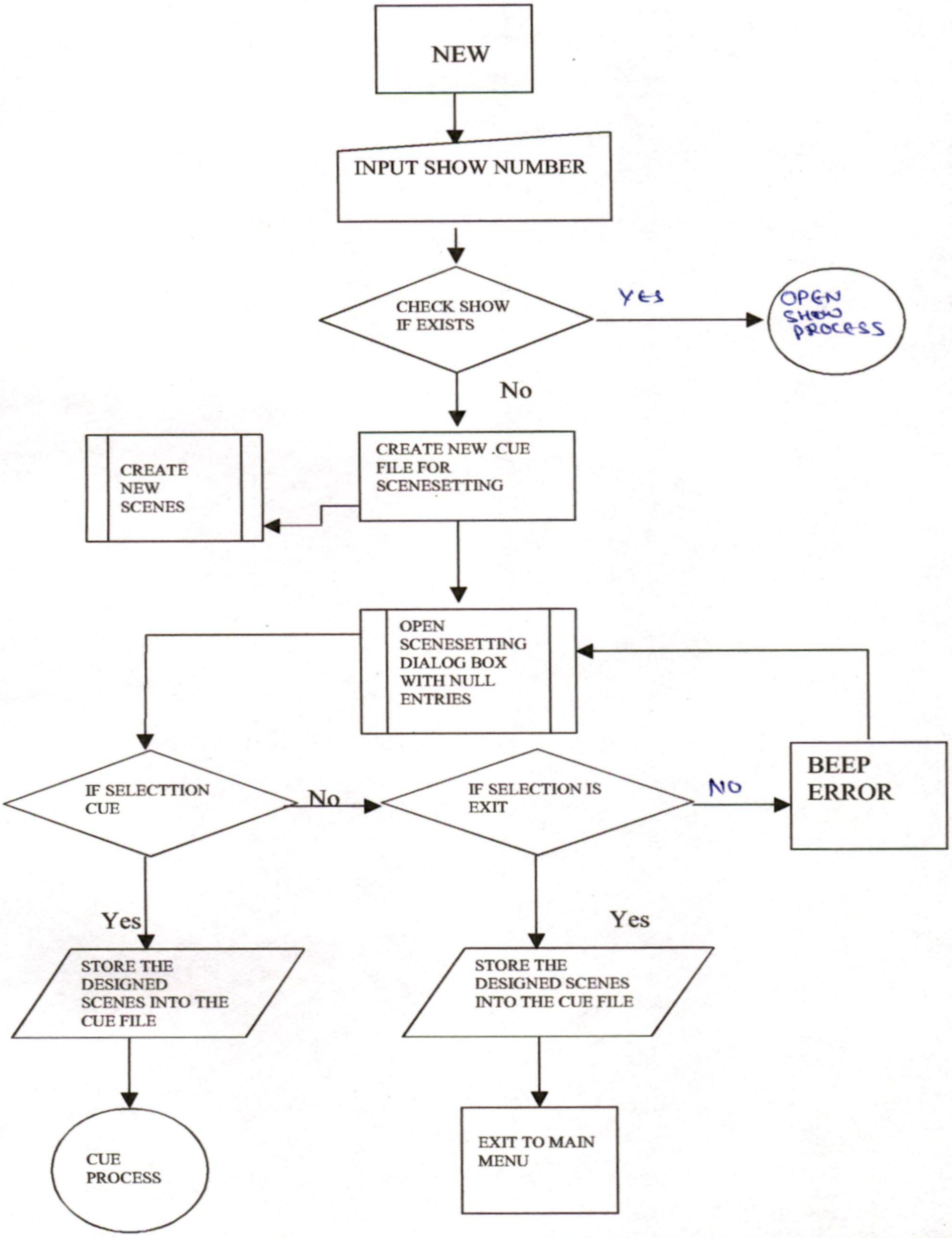
3.4.1 New show design for scene setting...

A New show design requires the following process:

- A show number – which is normally prompted to the user
- Creation of .CUE file on basis of this show number (for example SHOW1.CUE)
- Opening a default scene settings dialog box with null entries on zeroeth scene
- Allowing user to test individual scene if requested
- Save the changes after all the scenes have been created by user onto the corresponding .CUE file
- proceed onto create cues on basis of these scene or exit as requested by the user

The flowchart for the new show design for scene settings is on the following page.

Creating a new show in THEATRE VISION


NEW
INPUT SHOW NUMBER
CHECK SHOW IF EXISTS
YES
OPEN SHOW PROCESS
No
CREATE NEW .CUE FILE FOR SCENESETTING
CREATE NEW SCENES
OPEN SCENESETTING DIALOG BOX WITH NULL ENTRIES
IF SELECTTION CUE
No
IF SELECTION IS EXIT
NO
BEEP ERROR
Yes
Yes
STORE THE DESIGNED SCENES INTO THE CUE FILE
STORE THE DESIGNED SCENES INTO THE CUE FILE
CUE PROCESS
EXIT TO MAIN MENU

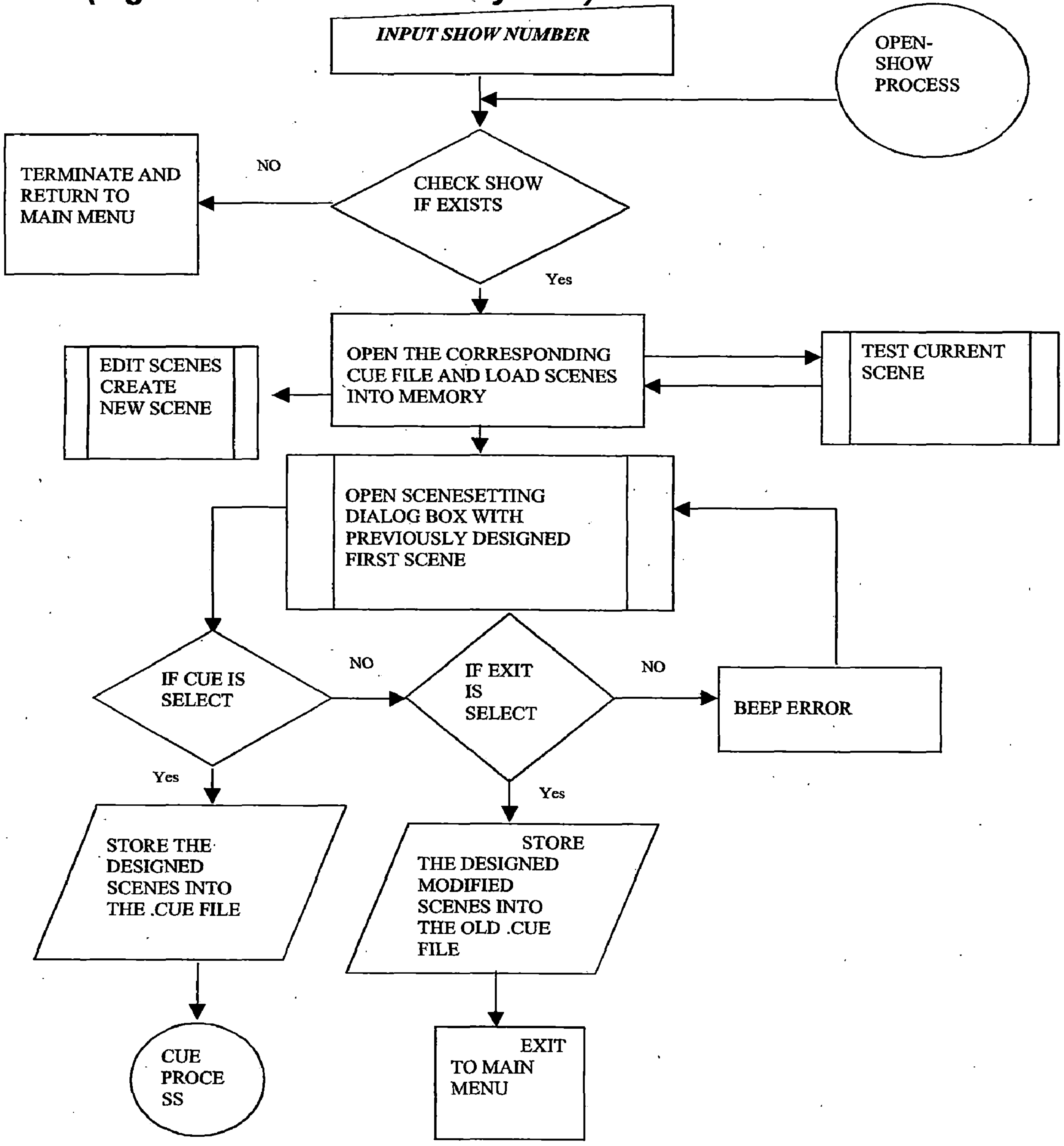
3.4.2 Editing/Opening an existing show for scene setting...!

Opening an existing show design requires the following process:

- The show number - which is normally the input from the user
- Checking weather it is a valid show (checking if corresponding .CUE files exist for requested show number)
- Opening the requested .CUE file
- Opening of scene setting dialog box with the entries stored for the zeroeth scene
- Allowing user to test the current scene if requested
- Save the changes after all the scenes have been newly created or edited by user onto the corresponding .CUE file
- proceed onto create cues on basis of these scenes or exit as requested by the user

The flowchart for Editing / Opening an existing show for scene settings is on the following page.

Editing / Opening an existing show in THEATRE VISION (Light and Sound control system)


INPUT SHOW NUMBER
OPEN-SHOW PROCESS
TERMINATE AND RETURN TO MAIN MENU
NO
CHECK SHOW IF EXISTS
Yes
EDIT SCENES CREATE NEW SCENE
OPEN THE CORRESPONDING CUE FILE AND LOAD SCENES INTO MEMORY
TEST CURRENT SCENE
OPEN SCENESETTING DIALOG BOX WITH PREVIOUSLY DESIGNED FIRST SCENE
IF CUE IS SELECT
NO
IF EXIT IS SELECT
NO
BEEP ERROR
Yes
Yes
STORE THE DESIGNED SCENES INTO THE .CUE FILE
STORE THE DESIGNED MODIFIED SCENES INTO THE OLD .CUE FILE
CUE PROCE SS
EXIT TO MAIN MENU

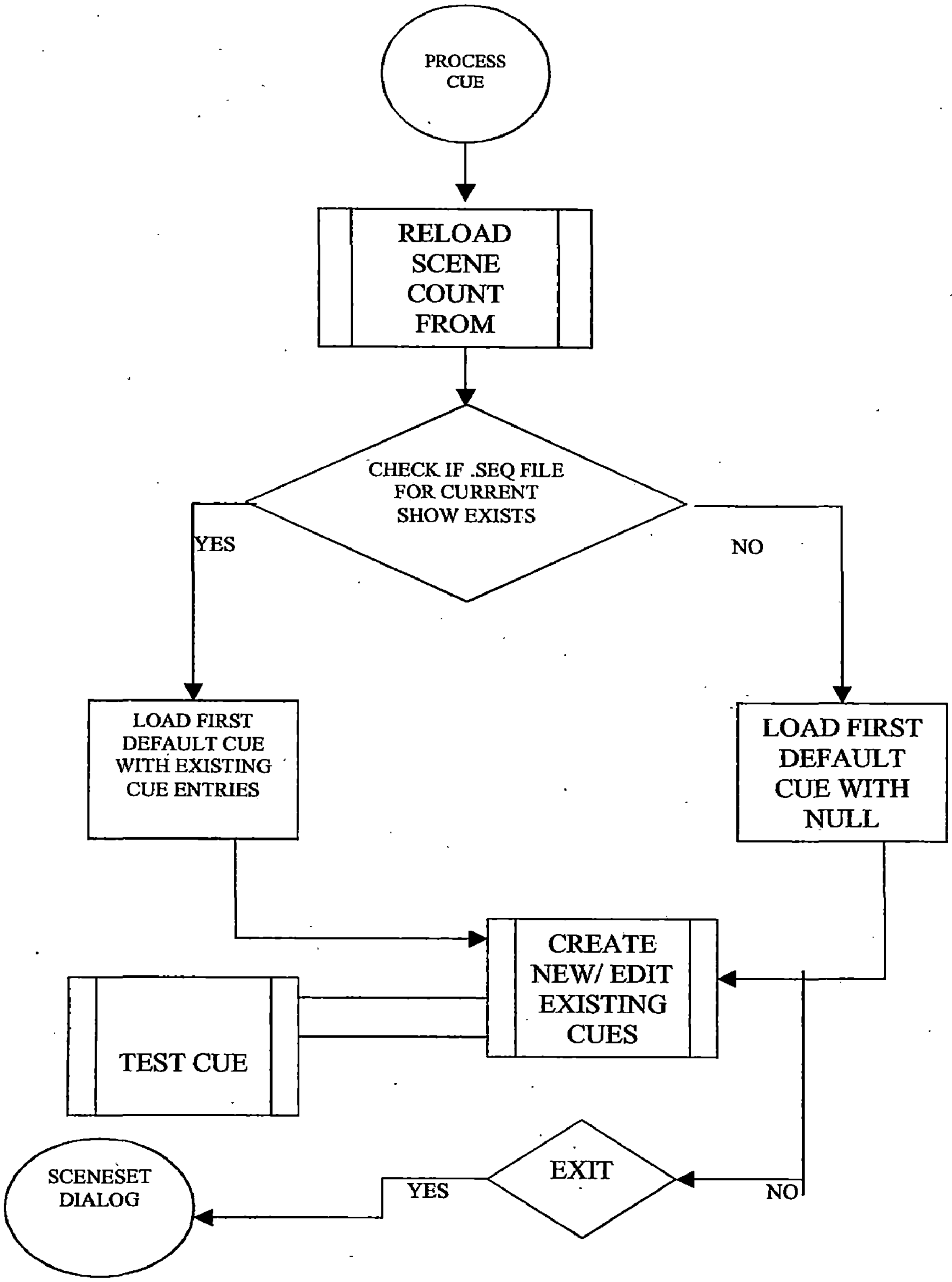
3.4.3 Process Designed scenes to make show CUEs... !

Designing cues for a complete show requires the following process:

- A show number – which is normally prompted by the user and the .CUE file is currently open for the requested show
- check if .SEQ file for the current show exist or not
- Creation of .SEQ file on basis of this shows number (for example SHOW1.SEQ) if the .SEQ has not been created earlier.
- Opening of default sequences dialog box with null entries on zeroeth cue if the .SEQ file has newly been created or else the stored entries for zeroeth cue are loaded
- Allow user to test individual cue if requested
- Save the changes after all the scenes have been created by user onto the corresponding .CUE file as user exits from sequences dialog (Note that it does not prompt you to save the changes-it does it on its own)
- Return back to the scene setting dialog box

The flowchart for the designing cue sequences is on the following page.

Process Designed Scenes To Make Show Cues


PROCESS CUE
RELOAD SCENE COUNT FROM
CHECK IF .SEQ FILE FOR CURRENT SHOW EXISTS
YES
NO
LOAD FIRST DEFAULT CUE WITH EXISTING CUE ENTRIES
LOAD FIRST DEFAULT CUE WITH NULL
CREATE NEW/ EDIT EXISTING CUES
TEST CUE
EXIT
YES
NO
SCENESET DIALOG

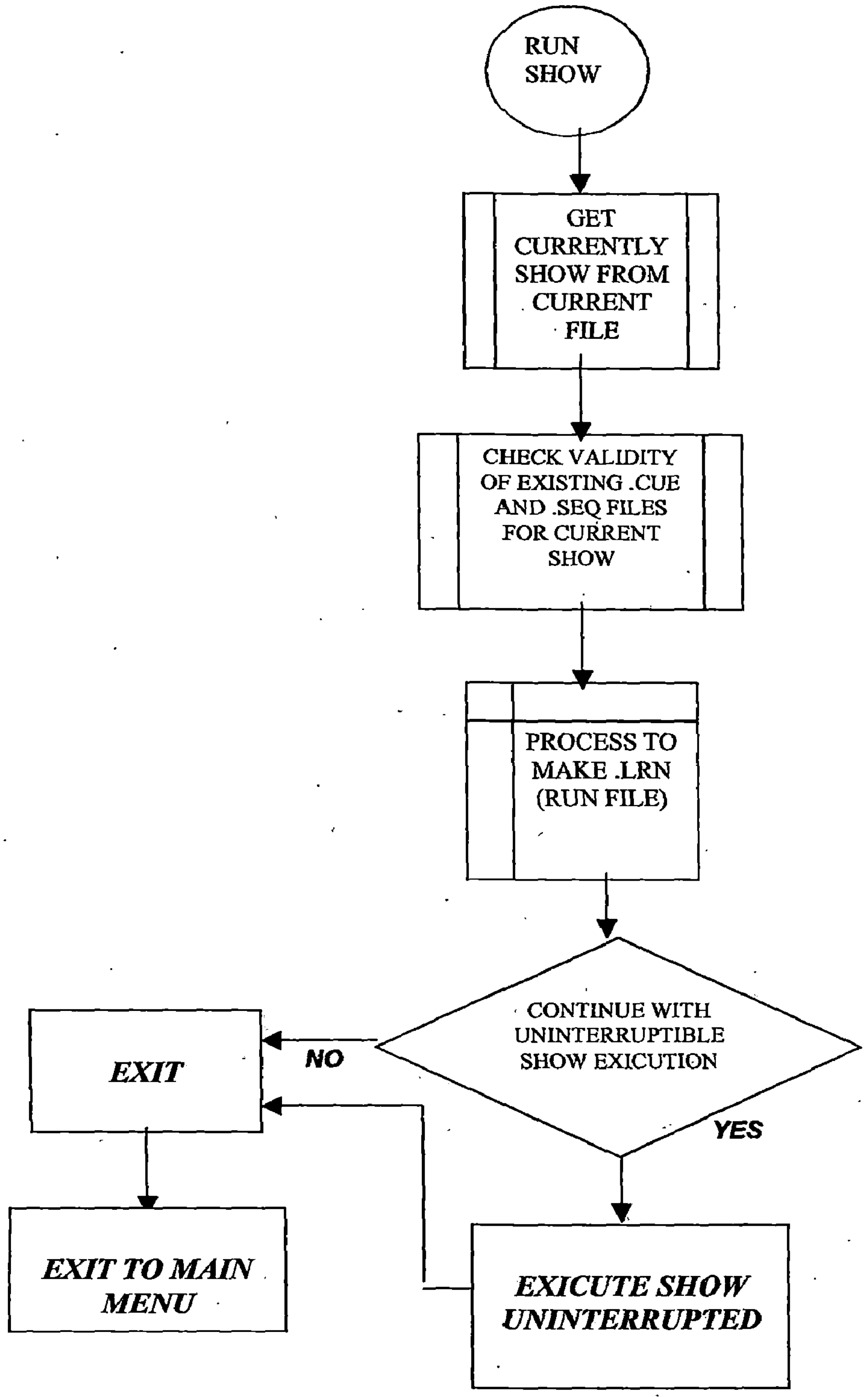
3.4.4 Final Run Show...... !

After designing cues and testing individually each of them. The system is now ready to independently run the current show in process and this require the following:

- A show number – which is availed from the CURRENT.FIL file which maintains count for the current show in process
- Check validity of existing .CUE and .SEQ files for consistency of data
- Process to create a runtime .LRN file which can allow an uninterrupted show execution
- Load clock and run show counters synchronized with the light data transfer
- Output data signal available on serial port asynchronously
- Prompt user after successful show completion to exit
- Return back to main menu

Flowchart on the next page

FINAL RUN SHOW IN THEATRE VISION


RUN SHOW
GET CURRENTLY SHOW FROM CURRENT FILE
CHECK VALIDITY OF EXISTING .CUE AND .SEQ FILES FOR CURRENT SHOW
PROCESS TO MAKE .LRN (RUN FILE)
CONTINUE WITH UNINTERRUPTIBLE SHOW EXICUTION
NO
YES
EXIT
EXIT TO MAIN MENU
EXICUTE SHOW UNINTERRUPTED

3.5 FILE FORMATS USED FOR STORING THE LIGHT DATA....!

The light and sound control uses two file formats .CUE and SEQ for storing the light data information which is in turn used at runtime to execute independently a light and sound show.

The .CUE file format used to store individual scene information can have 1000 scenes at the maximum and each scene can have fader settings for 512 dimmers. Each fader value can very between 0 and 255 with 0 being the minimum and 255 being the maximum.

The .SEQ file format used to store individual cue information can have 1,00,000 cues with each cue using only those number of scenes that corresponding .CUE file has got. Through it can process a maximum of 1000 scenes per cue with 4 effects in each case. Each scene can have either of four possible values – black out, fade in, fade out, and lock.

Black out means it wills remains at zero.

Fade in means it will grow from zero to the respective voltage.

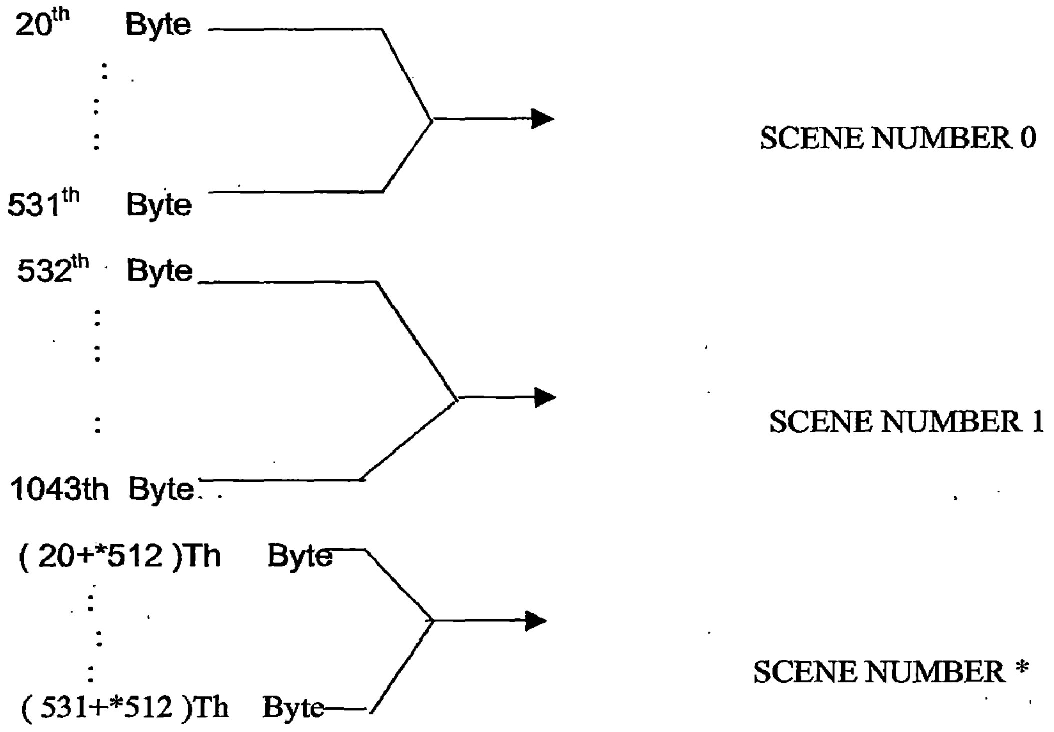
Fade out means it will decrease to zero from the respective voltage.

Lock means that it will remains at the same voltage for the defined period.

The two file formats are individually described bit wise on the following page.

3.6 SCENESETTING STORAGE FILE FORMAT(.CUE).

1ST	Bit	0	24	dimmers
2nd		1	512	dimmers
2nd	Byte			
3rd	Byte			
4th	Byte			
5th	Byte			
6th	Byte			CUE HEADER (PROTECTED)
7th	Byte			(INCLUDE AUTHOR NAME,
8th	Byte			COPYRIGHTS, VERSION, ETC.)
9th	Byte			
10th	Byte			
11th	Byte			
12th	Byte			
13th	Byte			
14th	Byte			
15th	Byte			
16th	Byte	Total number of scenes		
17th	Byte	Total MAX number of dimmers used(processed value)		
18th	Byte	File –Id number		
19th	Byte	File –Id number		



MAXIMUM LIMIT GOES UPTO 1000 SCENES

THIS IMPLIES THAT CUE FILE CAN BE AT THE MAXIMUM OF 512531(Approximately equal to 500 KB)

THIS IMPLIES THAT CUE FILE CAN BE AT THE MAXIMUM OF 50,051,855pproximately equal to 50.05 MB

MAXIMUM CUE SIZE : 500KB

MAXIMUM SEQ SIZE : 50.05 MB

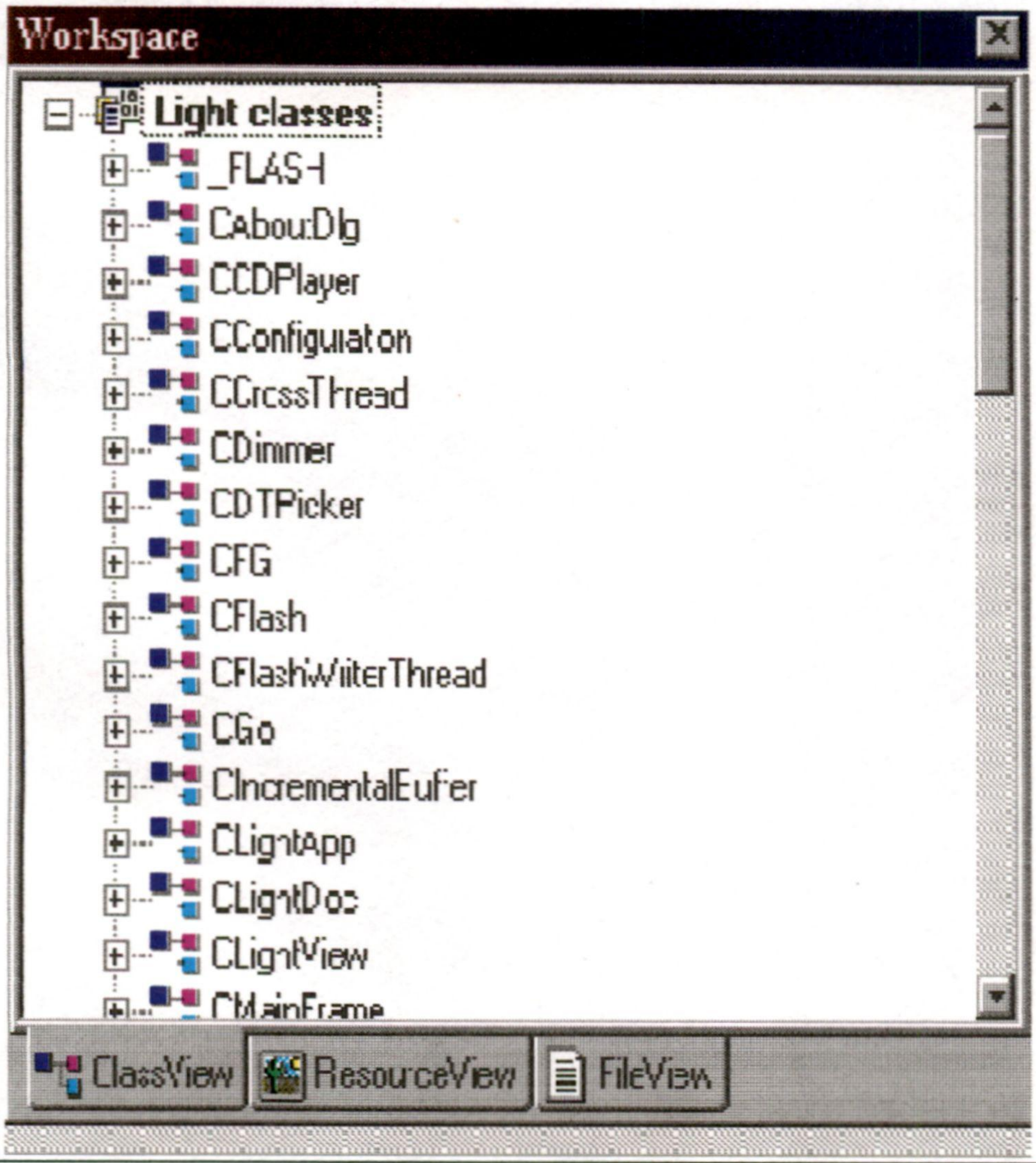
The maximum cue and seq file sizes can result a light and sound show working for a minimum of 88hours sing the maximum number of cues (1,00,000) and using the maximum number of scenes (1000) with the minimum time allotted for each cue (1 sec).

CHAPTER:4

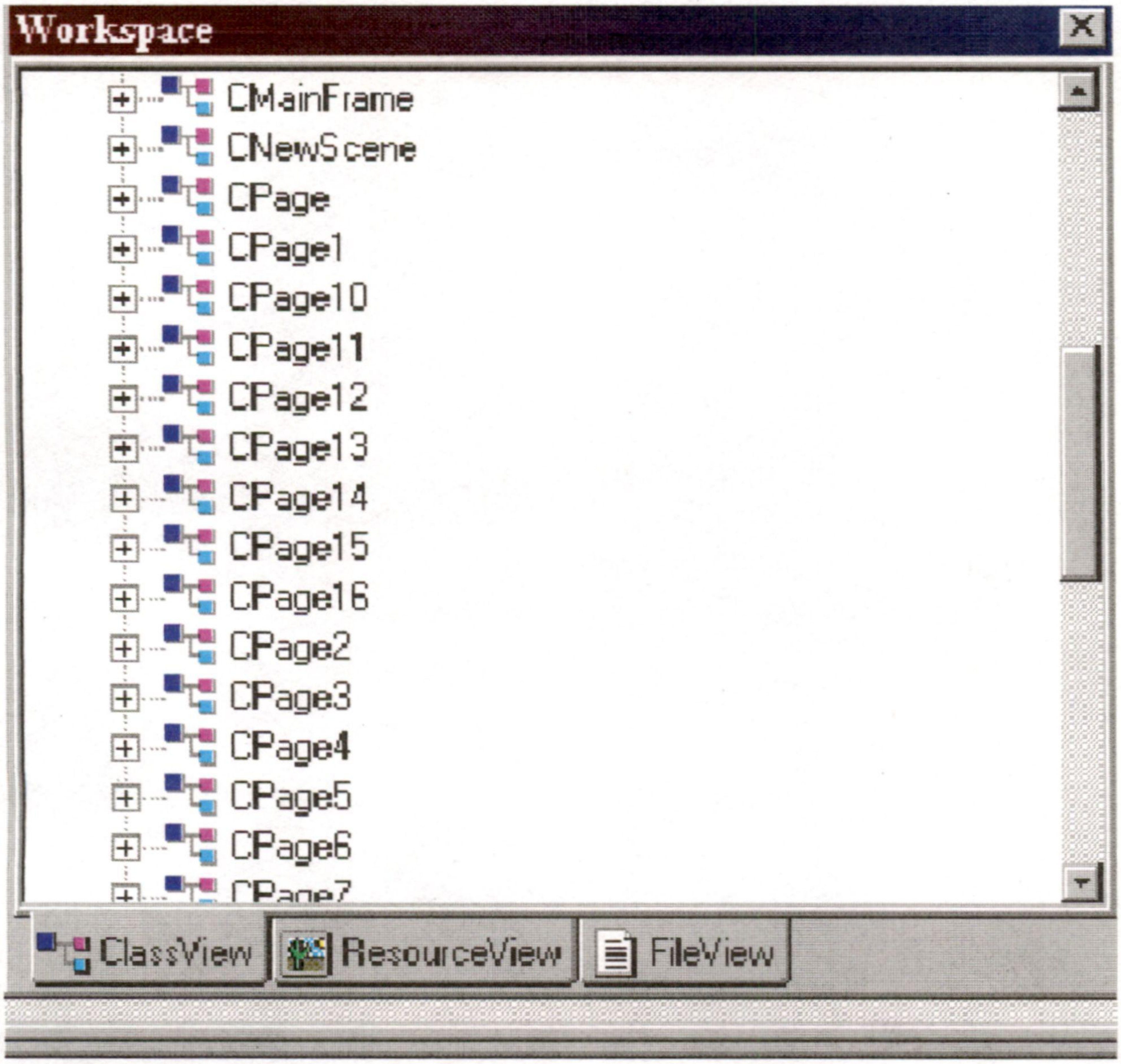
THE INFORMATION OF CLASSES

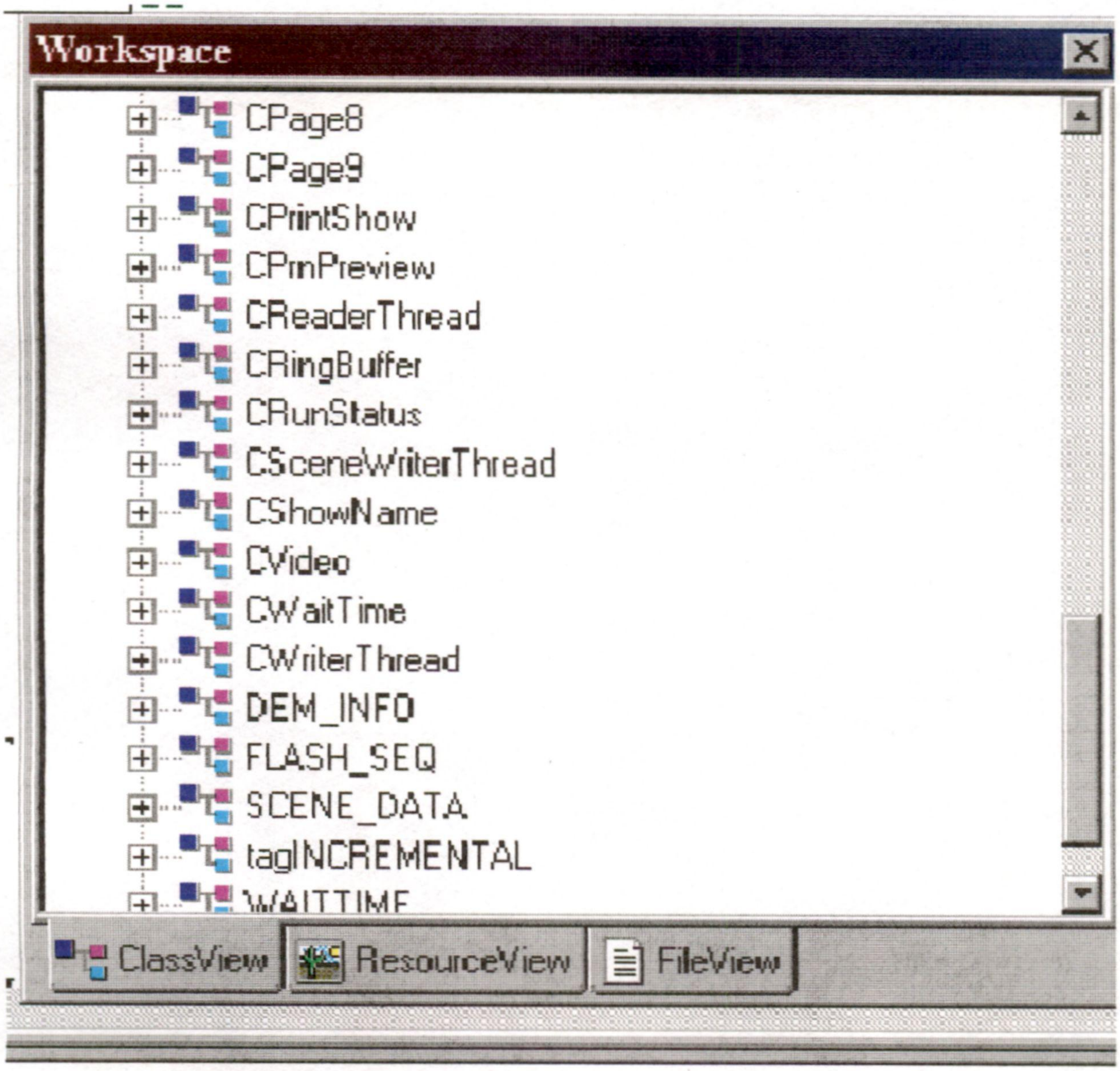
The coding of the THEATER VISION (Light and Sound control system) used visual C++ as programming language, Microsoft foundation classes ver 6.0 as shared DLLs, WIN 32 software development kit functions for those requirements which are not available through MFC 6.0 and finally inline assembly language for only interrupt handling whenever data needs to be transmitted to the dimmer controller.

Following classes have been used in the Project.



Light classes
- _FLASH
- CAboutDlg
- CCDPlayer
- CConfiguration
- CCrossThread
- CDimmer
- CDTPicker
- CFG
- CFlash
- CFlashWriterThread
- CGo
- CIncrementalBuffer
- CLightApp
- CLightDoc
- CLightView
- CMainFrame


Workspace
CMainFrame
CNewScene
CPage
CPage1
CPage10
CPage11
CPage12
CPage13
CPage14
CPage15
CPage16
CPage2
CPage3
CPage4
CPage5
CPage6
CPage7
ClassView
ResourceView
FileView



Workspace

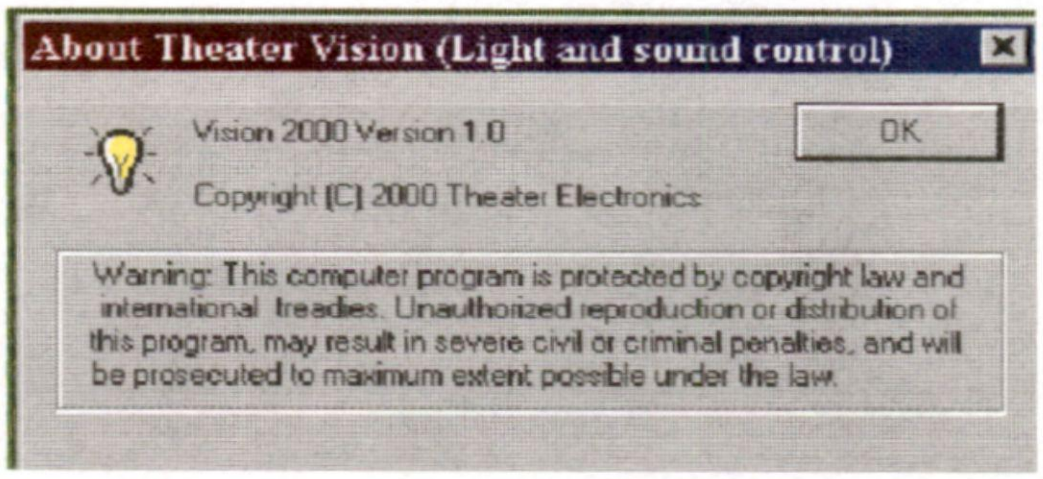
CPage8
CPage9
CPrintShow
CPrnPreview
CReaderThread
CRingBuffer
CRunStatus
CSceneWriterThread
CShowName
CVideo
CWaitTime
CWriterThread
DEM_INFO
FLASH_SEQ
SCENE_DATA
tagINCREMENTAL
WAITTIME

Class Name: CAboutDlg

CAboutDlg is the Dialog Box, which displays information about the user's copy of THREATOR VISION, including the version number and the copyright.

Base Class for CAboutDlg is CDialog and most class members have been used.

Returns : NULL

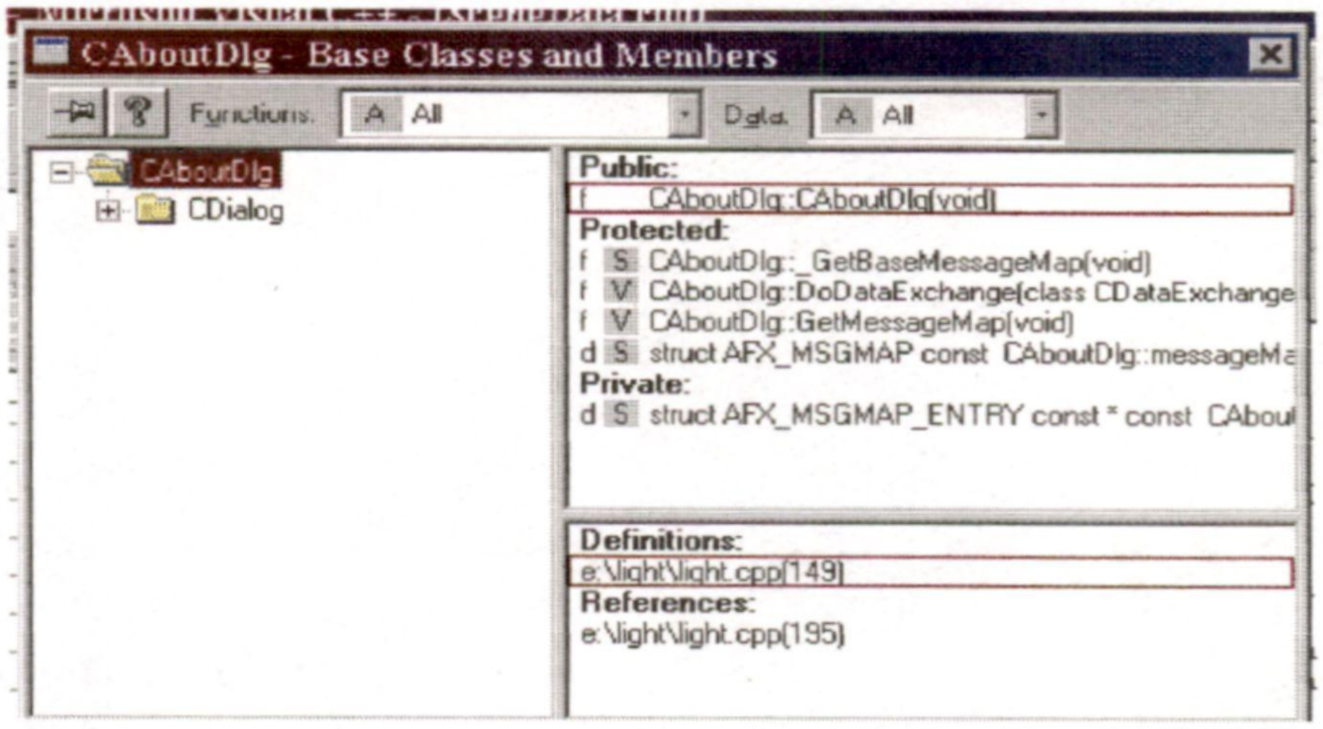


About Theater Vision (Light and sound control)

Vision 2000 Version 1.0

Copyright (C) 2000 Theater Electronics

Warning: This computer program is protected by copyright law and international treadies. Unauthorized reproduction or distribution of this program, may result in severe civil or criminal penalties, and will be prosecuted to maximum extent possible under the law.



CAboutDlg - Base Classes and Members

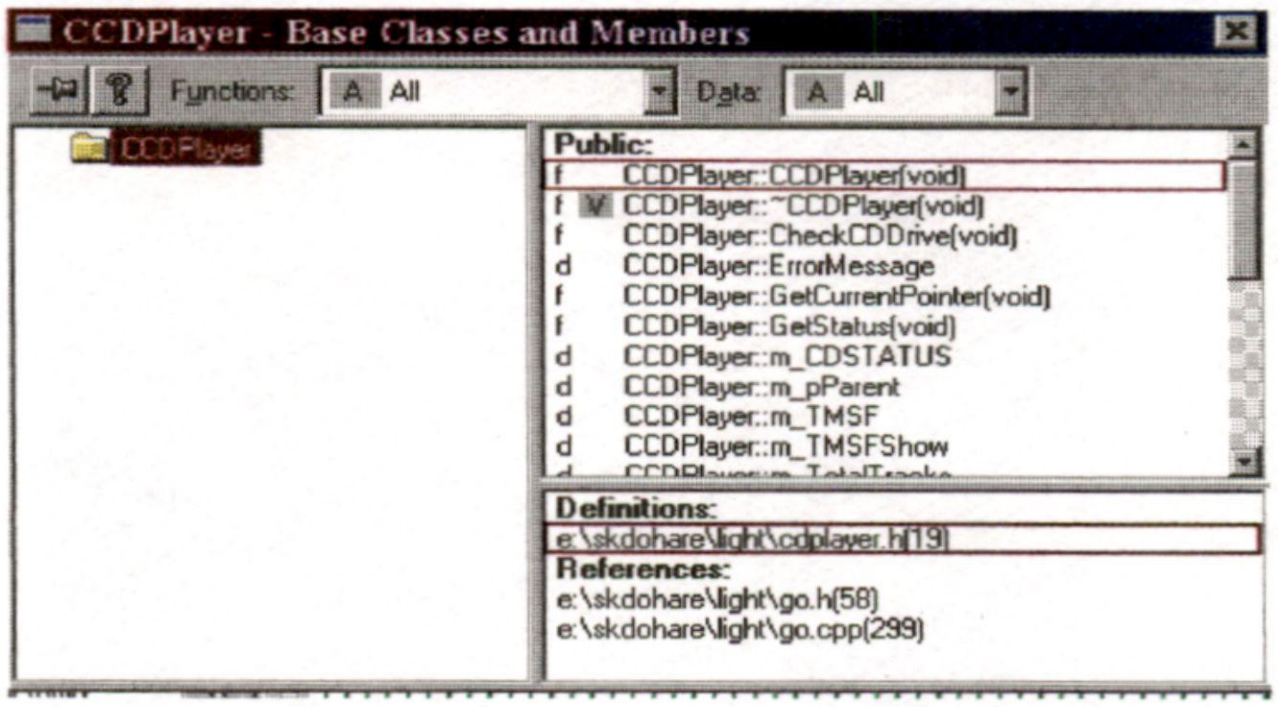
Functions: All   Data: All

```
CAboutDlg
  CDialog

Public:
f     CAboutDlg::CAboutDlg(void)
Protected:
f  S  CAboutDlg::_GetBaseMessageMap(void)
f  V  CAboutDlg::DoDataExchange(class CDataExchange
f  V  CAboutDlg::GetMessageMap(void)
d  S  struct AFX_MSGMAP const  CAboutDlg::messageMa
Private:
d  S  struct AFX_MSGMAP_ENTRY const * const  CAbou

Definitions:
e:\light\light.cpp(149)
References:
e:\light\light.cpp(195)
```

Class Name: CCDPlayer

Is the Dialog Box which displays information about the CD which you have to run during the Light Show. It displays the track no.

It returns the track no. to play the songs.



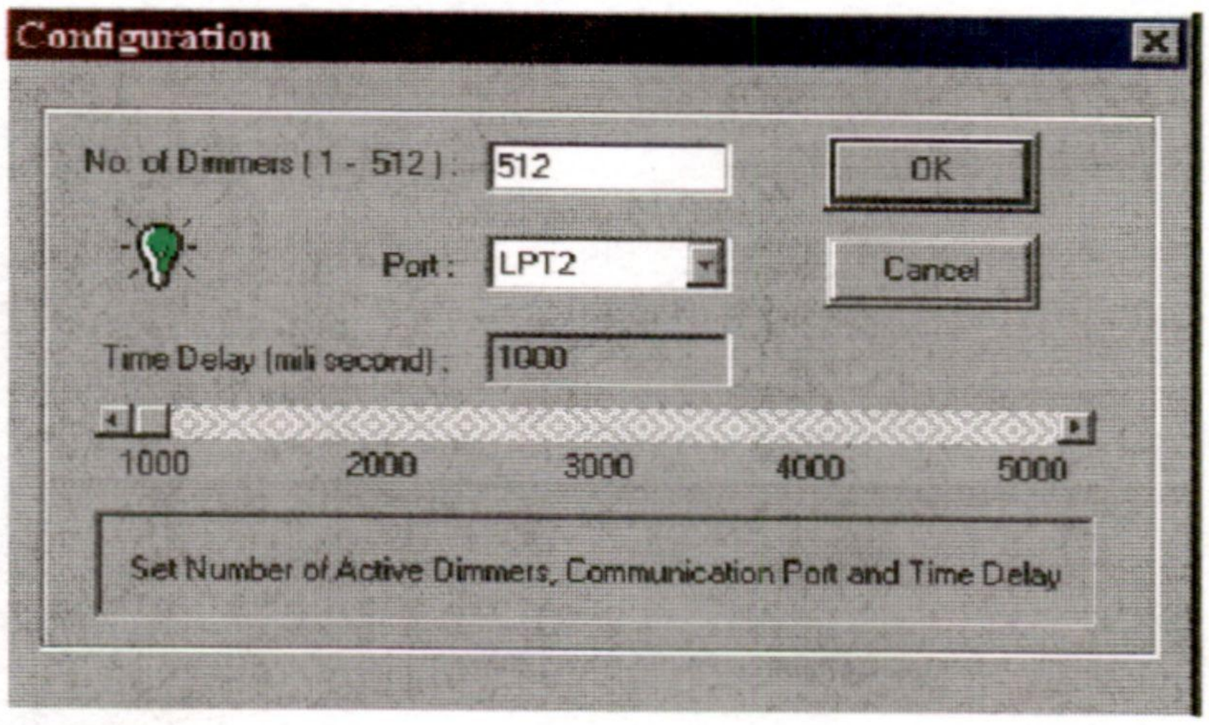
```
Public:
f      CCDPlayer::CCDPlayer(void)
f  V   CCDPlayer::~CCDPlayer(void)
f      CCDPlayer::CheckCDDrive(void)
d      CCDPlayer::ErrorMessage
f      CCDPlayer::GetCurrentPointer(void)
f      CCDPlayer::GetStatus(void)
d      CCDPlayer::m_CDSTATUS
d      CCDPlayer::m_pParent
d      CCDPlayer::m_TMSF
d      CCDPlayer::m_TMSFShow
Definitions:
e:\skdohare\light\cdplayer.h(19)
References:
e:\skdohare\light\go.h(58)
e:\skdohare\light\go.cpp(299)
```

Class Name : CConfiguration

CConfiguration is the Dialog Box Class which contains the information about the range of Active Dimmers used and the port information on which the DMX device is connected.
It also contains the information about the delay time in milliseconds the range is 1000 – 5000

Base Class for Configuration is CDialog and most class members have been used.

Protected Member Functions used the OnOK() and OnClose() member functions besides InitDialog().

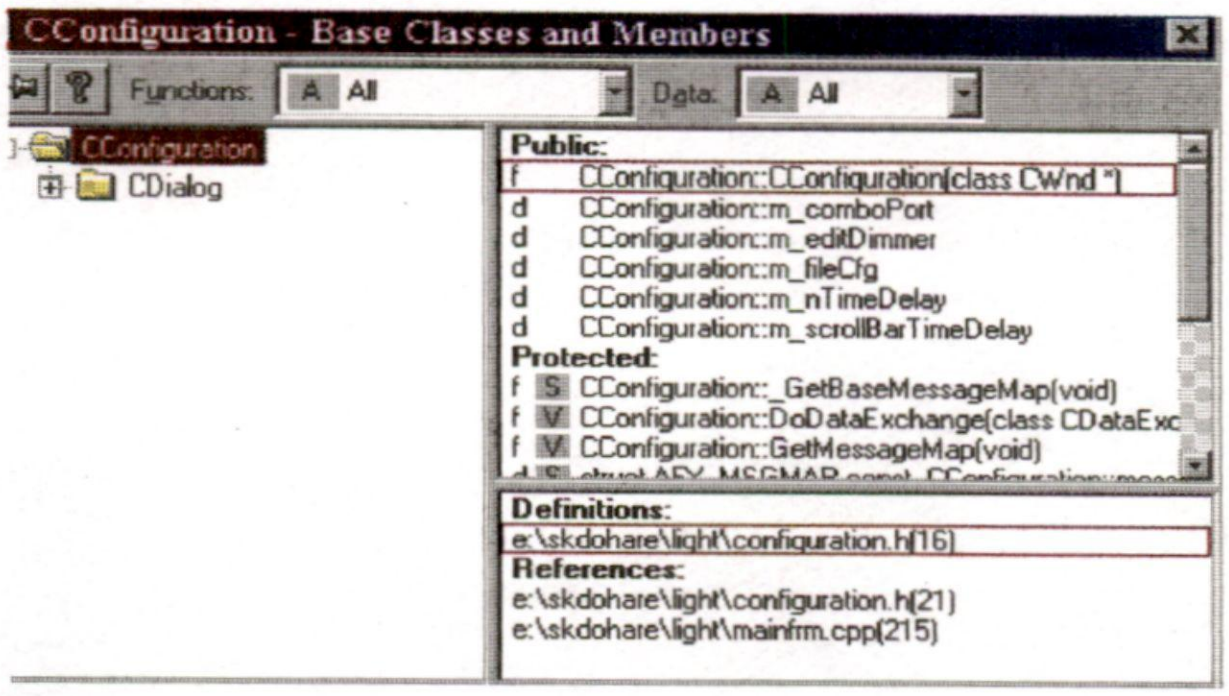


No. of Dimmers [1 - 512] : 512

Port : LPT2

Time Delay (mili second) : 1000

Set Number of Active Dimmers, Communication Port and Time Delay



```
Public:
f     CConfiguration::CConfiguration(class CWnd *)
d     CConfiguration::m_comboPort
d     CConfiguration::m_editDimmer
d     CConfiguration::m_fileCfg
d     CConfiguration::m_nTimeDelay
d     CConfiguration::m_scrollBarTimeDelay
Protected:
f  S  CConfiguration::_GetBaseMessageMap(void)
f  V  CConfiguration::DoDataExchange(class CDataExc
f  V  CConfiguration::GetMessageMap(void)

Definitions:
e:\skdohare\light\configuration.h(16)
References:
e:\skdohare\light\configuration.h(21)
e:\skdohare\light\mainfrm.cpp(215)
```

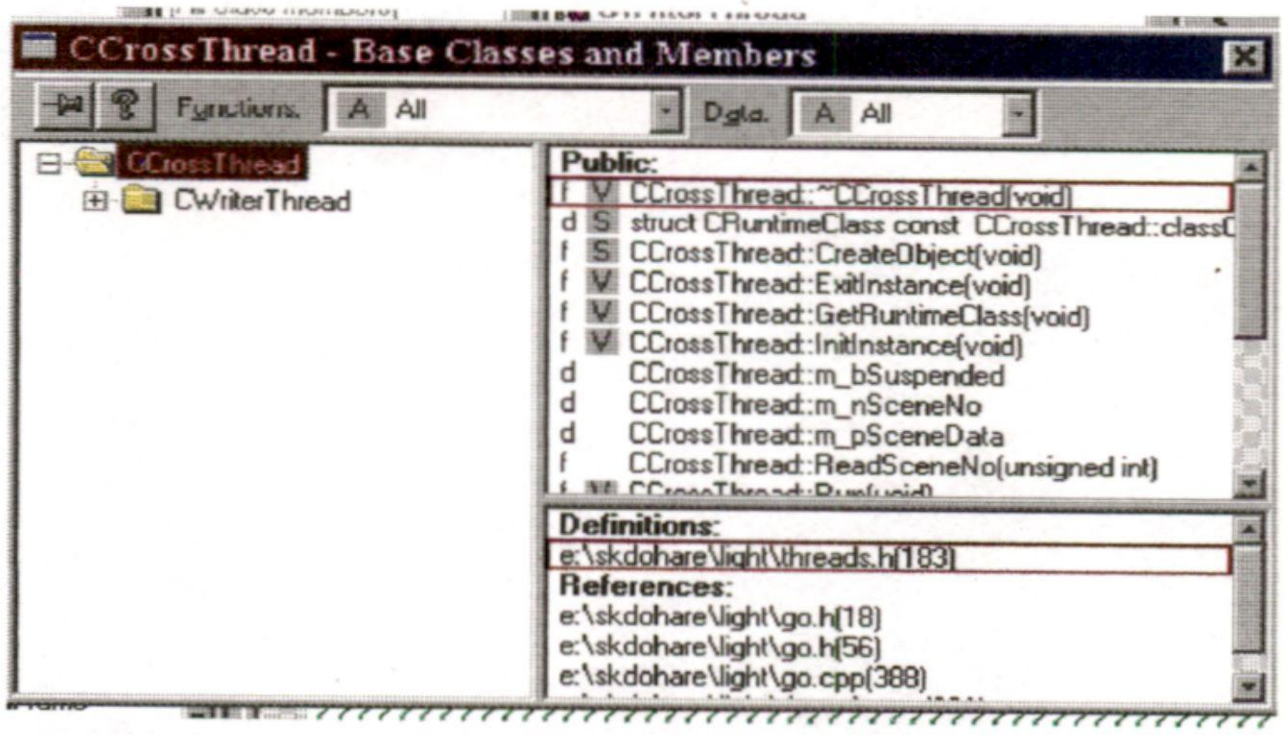
Class Name: CCrossThread

CCrossThread is the Dialog Box Class, which contains the information about Scene No. Which have to be connected with the current scene and it has to be decreased light.

Protected constructor used by dynamic creation

Base Class of CCrossThread class is CWriterThread .

It returns the scene no. Which have to be decreased light.

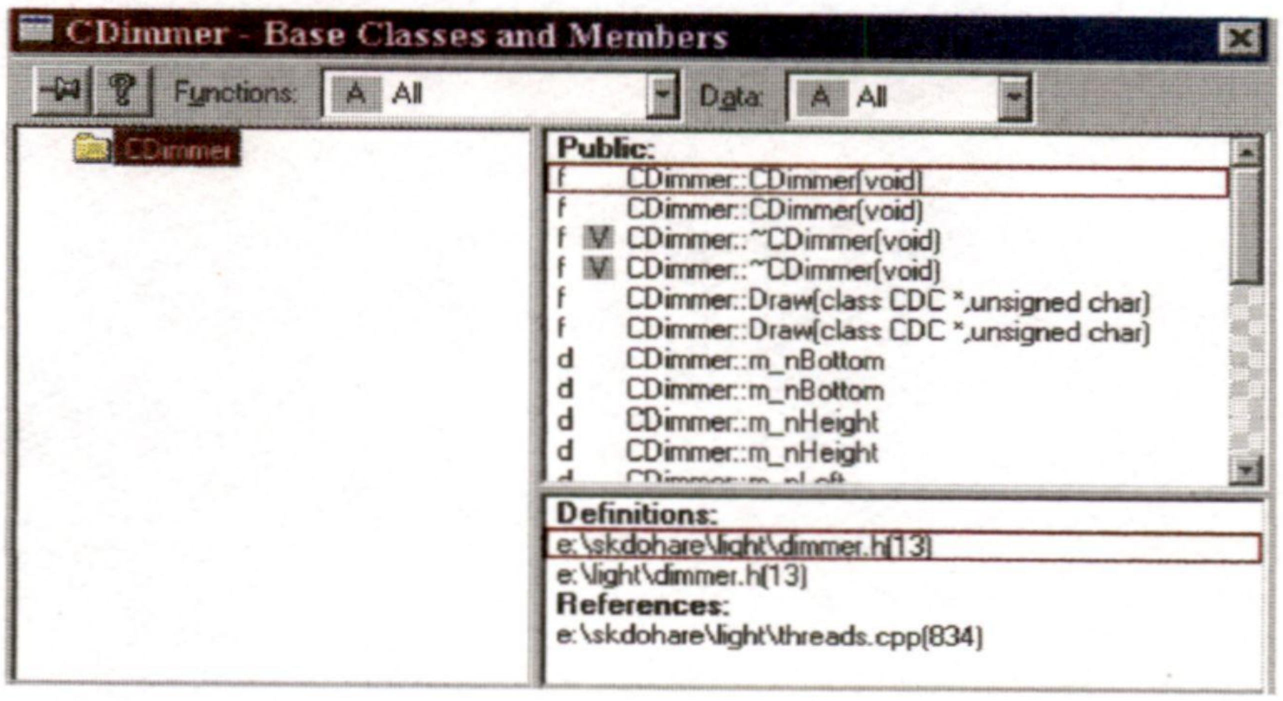


```
CCrossThread - Base Classes and Members
Functions: A All          Data: A All

CCrossThread
  CWriterThread

Public:
f V CCrossThread::~CCrossThread(void)
d S struct CRuntimeClass const CCrossThread::classC
f S CCrossThread::CreateObject(void)
f V CCrossThread::ExitInstance(void)
f V CCrossThread::GetRuntimeClass(void)
f V CCrossThread::InitInstance(void)
d   CCrossThread::m_bSuspended
d   CCrossThread::m_nSceneNo
d   CCrossThread::m_pSceneData
f   CCrossThread::ReadSceneNo(unsigned int)

Definitions:
e:\skdohare\light\threads.h(183)
References:
e:\skdohare\light\go.h(18)
e:\skdohare\light\go.h(56)
e:\skdohare\light\go.cpp(388)
```

Class Name: CDimmers

CDimmer is the Class, which contains the information about the creation of show, which glows the dimmers. First it create Black color dimmer and when it is selected with value it glows with Red color.

Returns : NULL

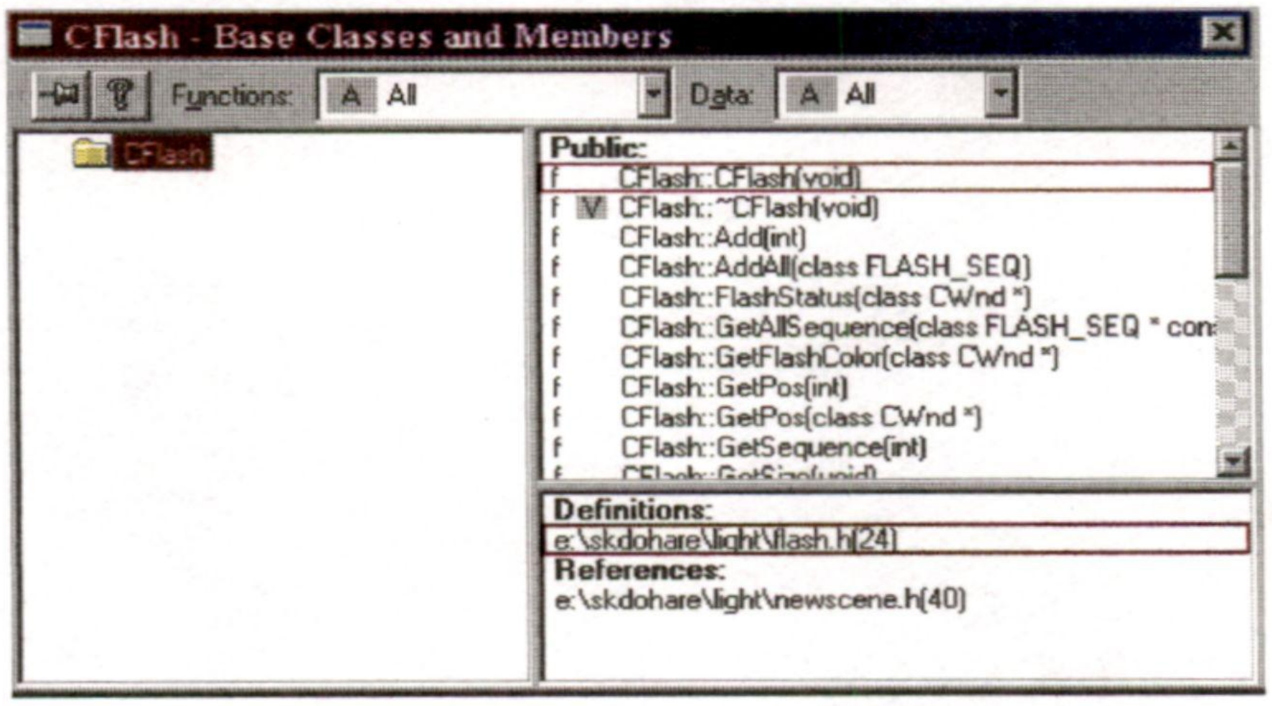


```
CDimmer - Base Classes and Members
Functions: A All          Data: A All

CDimmer

Public:
f   CDimmer::CDimmer(void)
f   CDimmer::CDimmer(void)
f V CDimmer::~CDimmer(void)
f V CDimmer::~CDimmer(void)
f   CDimmer::Draw(class CDC *,unsigned char)
f   CDimmer::Draw(class CDC *,unsigned char)
d   CDimmer::m_nBottom
d   CDimmer::m_nBottom
d   CDimmer::m_nHeight
d   CDimmer::m_nHeight

Definitions:
e:\skdohare\light\dimmer.h(13)
e:\light\dimmer.h(13)
References:
e:\skdohare\light\threads.cpp(834)
```

Class Name: CFlash

CFlash is the class which contains the information about the flash run. How it runs and It uses CNewScene and CCriticalSection class and use following functions:

AddAll(), GetSize(), PutAllSequence, GetAllSequence
ResetFlash(), IsFlashWnd, FlashStatus, GetPos, Add,
SetOwnerShip, SetFlash, GetFlashColor .

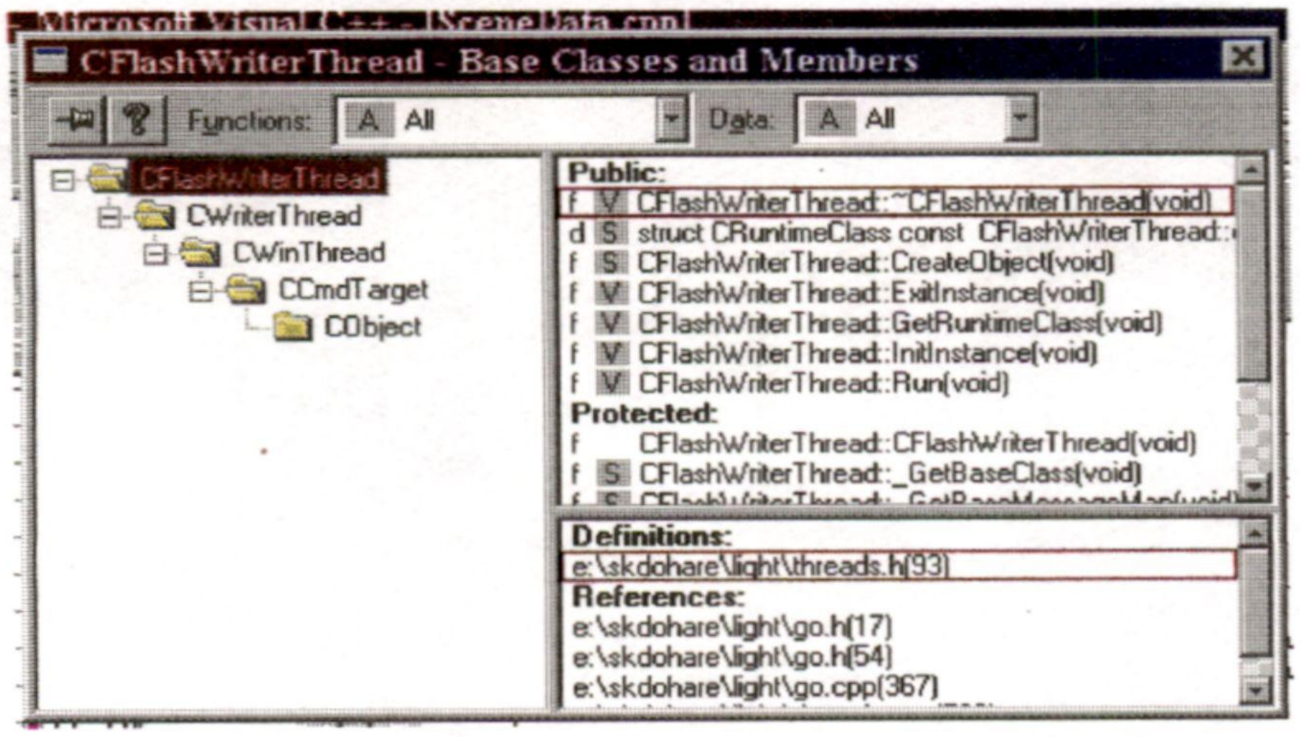


```
Public:
f     CFlash::CFlash(void)
f  V  CFlash::~CFlash(void)
f     CFlash::Add(int)
f     CFlash::AddAll(class FLASH_SEQ)
f     CFlash::FlashStatus(class CWnd *)
f     CFlash::GetAllSequence(class FLASH_SEQ * con
f     CFlash::GetFlashColor(class CWnd *)
f     CFlash::GetPos(int)
f     CFlash::GetPos(class CWnd *)
f     CFlash::GetSequence(int)

Definitions:
e:\skdohare\light\flash.h(24)
References:
e:\skdohare\light\newscene.h(40)
```

Class Name: CFlashWriterThread

This class uses write pattern of flash. .it declare the full 255 value in the dimmers value.

The Base Class of CFlashWriterThread is CWriterThread



```
Public:
f  V  CFlashWriterThread::~CFlashWriterThread(void)
d  S  struct CRuntimeClass const CFlashWriterThread::
f  S  CFlashWriterThread::CreateObject(void)
f  V  CFlashWriterThread::ExitInstance(void)
f  V  CFlashWriterThread::GetRuntimeClass(void)
f  V  CFlashWriterThread::InitInstance(void)
f  V  CFlashWriterThread::Run(void)
Protected:
f     CFlashWriterThread::CFlashWriterThread(void)
f  S  CFlashWriterThread::_GetBaseClass(void)

Definitions:
e:\skdohare\light\threads.h(93)
References:
e:\skdohare\light\go.h(17)
e:\skdohare\light\go.h(54)
e:\skdohare\light\go.cpp(367)
```

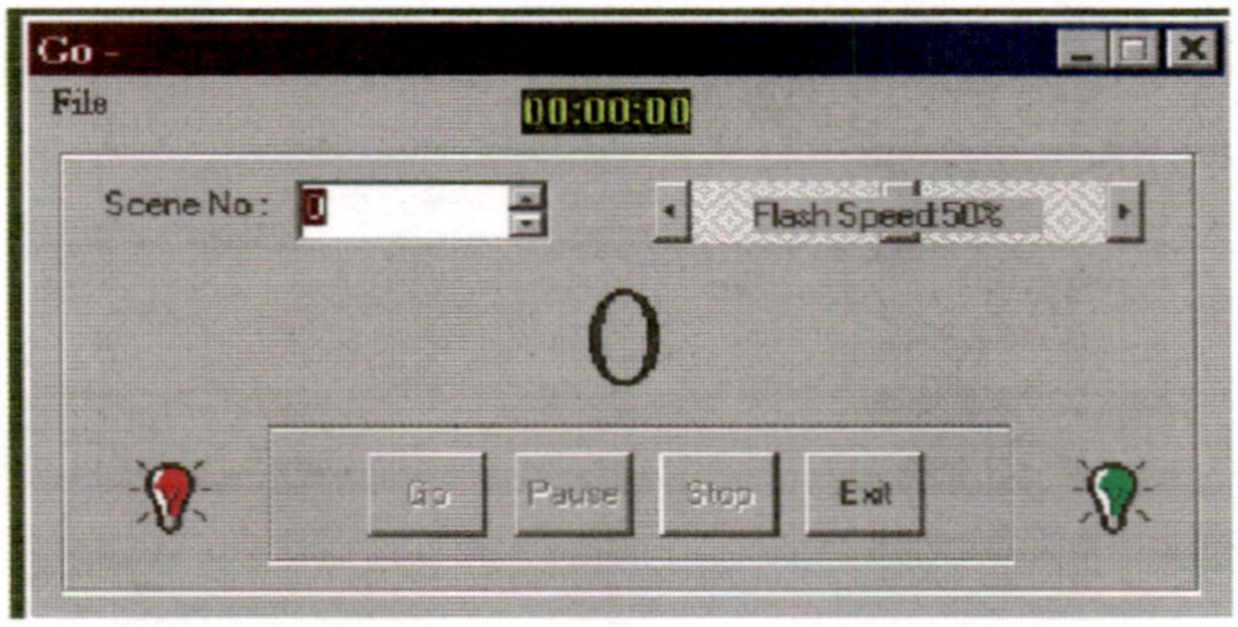
Class Name: CGO

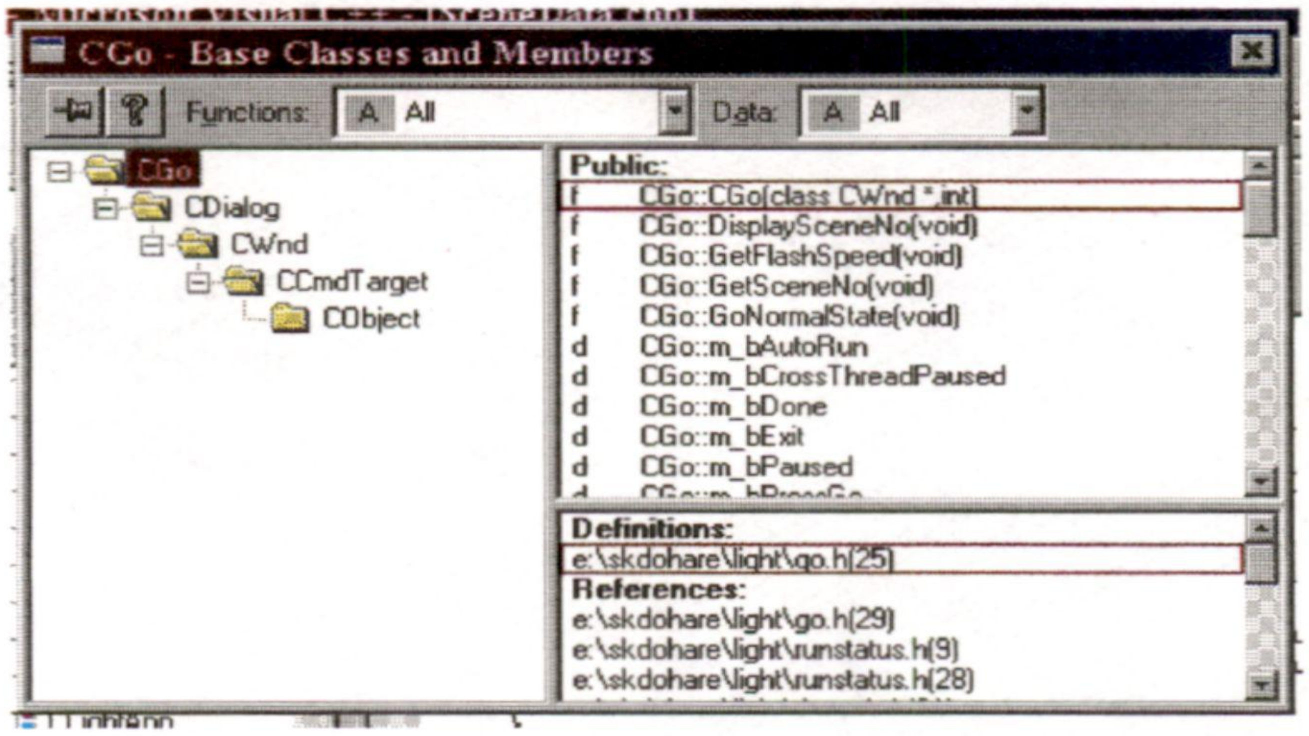
CGO is the Dialog Box which displays information about the .cue file which have to be run for the show. The show no. is availed from the current .File file. The .cue and .seq files are checked for validation. A runtime file .LRN which has filtered each cue data corresponding to 521 dimmers associated with time is created It display the running show. And uses threads classes for activation of dimmers.

Before activation of this dialog the clock is activated asynchronously.

Base Class for CGO is CDialog and most class members have been used.

Protected Member Functions used the OnClose() , Open(), Pause(), Go(), Stop() member functions besides InitDialog().




CGo - Base Classes and Members

Functions: A All   Data: A All

```
⊟ CGo
   ⊟ CDialog
      ⊟ CWnd
         ⊟ CCmdTarget
            └ CObject
```

```
Public:
f      CGo::CGo(class CWnd *,int)
f      CGo::DisplaySceneNo(void)
f      CGo::GetFlashSpeed(void)
f      CGo::GetSceneNo(void)
f      CGo::GoNormalState(void)
d      CGo::m_bAutoRun
d      CGo::m_bCrossThreadPaused
d      CGo::m_bDone
d      CGo::m_bExit
d      CGo::m_bPaused
```

```
Definitions:
e:\skdohare\light\go.h(25)
References:
e:\skdohare\light\go.h(29)
e:\skdohare\light\runstatus.h(9)
e:\skdohare\light\runstatus.h(28)
```

Class Name: CLightApp

The main application class CLightApp derived form CWinApp which encaplualtes the initialization, running, and termination of a application for Windows. An application built on the framework must have one (and only one) object of a class derived from CWinApp . this object is constructed our application's primary thread of execution. Using Win32 API functions, secondary threads of execution are also created.

Our framework application has a WinMain function . WinMain performs standard services such as registering window classes. Then it calls member functions of the application object to initialize and run appoication.

To initialize the application, WinMain calls our application object's InitApplication and InitInstance member fuctions. To run the application's message loop, WinMain calls the Run member function. On termination , WinMain calls the applications object's ExitInstance member function.

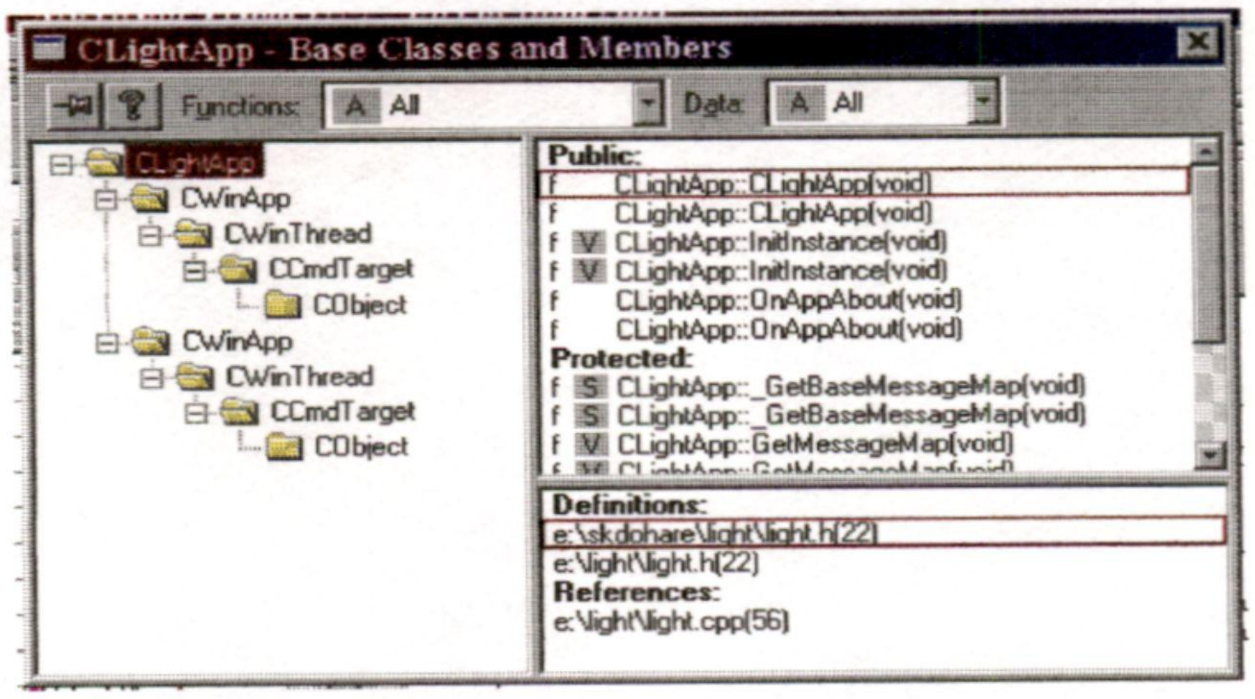
Figure : Sequence of Execution

WinMain Standard function supplied by framework

Calls → InitInstance initializes current instance of the application

Calls → **Run** Runs the message loop and OnIdle

Calls → ExitInstance cleans up after the application



Public:
f CLightApp::CLightApp(void)
f CLightApp::CLightApp(void)
f V CLightApp::InitInstance(void)
f V CLightApp::InitInstance(void)
f CLightApp::OnAppAbout(void)
f CLightApp::OnAppAbout(void)
Protected:
f S CLightApp::_GetBaseMessageMap(void)
f S CLightApp::_GetBaseMessageMap(void)
f V CLightApp::GetMessageMap(void)

Definitions:
e:\skdohare\light\light.h(22)
e:\light\light.h(22)
References:
e:\light\light.cpp(56)

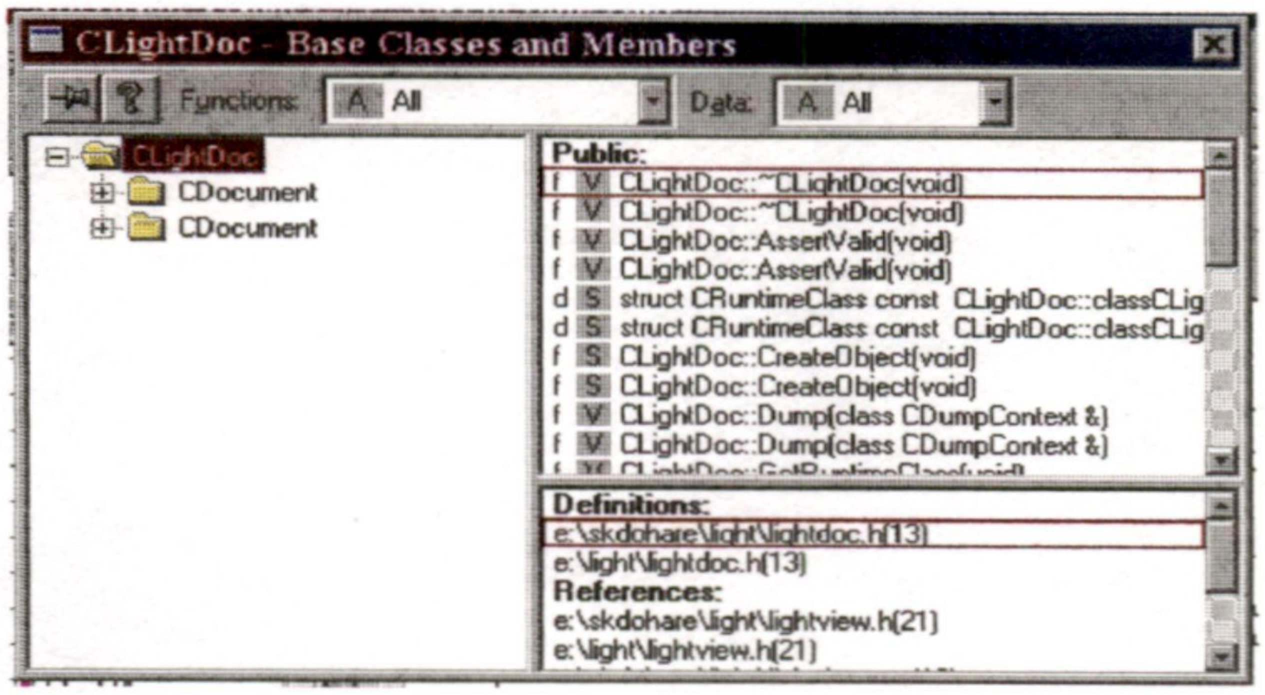
Class Name: CLightDoc

CLightDoc derived from the CDocument class Provides the basic functionality for user defined document classes. A document represents the unit of data that the user typically opens with the File Open command and saves with File Save command.

CDoucment supports standard operations such as creating a document, loading it, and saving it. The framework manipulates using the interface defined by CDocument.

So Light application can support more than one type of document; for example, an application might support both spreadsheets and text documents. Each type of document has an associated document template; the document template specifies what resources (for example, menu, icon, accelerator table) are used for that type of document . each document contains a pointer to its associated CDocTemplate object.

Users interact with a document through the CView object associated with it. A view renders an image of the document in a frame window and interprets user input as operations on the document . a document can have multiple views associated with it. When the user opens a



```
Public:
f V CLightDoc::~CLightDoc(void)
f V CLightDoc::~CLightDoc(void)
f V CLightDoc::AssertValid(void)
f V CLightDoc::AssertValid(void)
d S struct CRuntimeClass const  CLightDoc::classCLig
d S struct CRuntimeClass const  CLightDoc::classCLig
f S CLightDoc::CreateObject(void)
f S CLightDoc::CreateObject(void)
f V CLightDoc::Dump(class CDumpContext &)
f V CLightDoc::Dump(class CDumpContext &)
```

```
Definitions:
e:\skdohare\light\lightdoc.h(13)
e:\light\lightdoc.h(13)
References:
e:\skdohare\light\lightview.h(21)
e:\light\lightview.h(21)
```

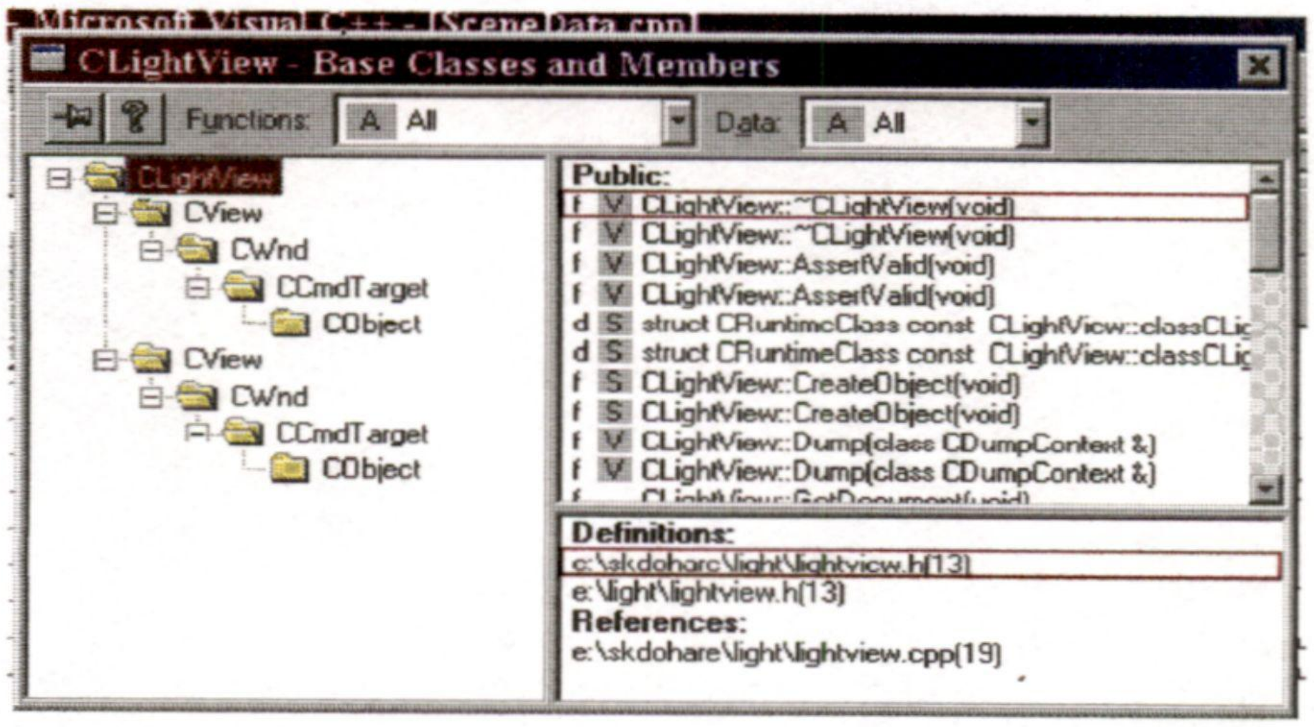
window on a document, the framework creates a view and attaches it to document. The document template specifies what type of view and frame window are used to display each of document. Through LIGHT does not explore much of the CDocument properties.

Class Name: CLightView

The CLightView class provides the basic functionality for user-defined view classes. A view is attached to a document and acts as an intermediately between the document and the user: the view renders an image of the document on the screen or printer and interprets user input as operations upon the document.

A view is a child of a frame window. More than one view can share a frame window, as in the case of a splitter window. The relationship between a view class, and frame window class, and a document class is established by a CDocTemplate object. When the user opens a new window or splits an existing one, the framework constructs a new view and attaches it to the document.

A view may be responsible for handling several different types of input, such as keyboard input. Mouse input or input via drag-and drop, as well as commands from menus, toolbars or scroll bars. A view receives commands forwarded by its frame window. If the view does not handle a given command, it forwards the commands to its associated document. Like all command targets, a view handles messages via a message map.


Microsoft Visual C++ - [SceneData.cpp]
CLightView - Base Classes and Members
Functions: A All
Data: A All

```
CLightView
  CView
    CWnd
      CCmdTarget
        CObject
  CView
    CWnd
      CCmdTarget
        CObject
```

```
Public:
f V CLightView::~CLightView(void)
f V CLightView::~CLightView(void)
f V CLightView::AssertValid(void)
f V CLightView::AssertValid(void)
d S struct CRuntimeClass const CLightView::classCLig
d S struct CRuntimeClass const CLightView::classCLig
f S CLightView::CreateObject(void)
f S CLightView::CreateObject(void)
f V CLightView::Dump(class CDumpContext &)
f V CLightView::Dump(class CDumpContext &)
f   CLightView::GetDocument(void)
Definitions:
c:\skdohare\light\lightview.h(13)
e:\light\lightview.h(13)
References:
e:\skdohare\light\lightview.cpp(19)
```

Class Name: CMainFrame

The CMainFrame class derived from CMDIFrameWnd class provides the functionality of a window multiple document interface (MDI) frame window, along with members for managing the window.

The MDI frame window is created by calling the Create or LoadFrame member function of CFrameWnd.

The MDI frame window manages the MDICLIENT window, repositioning it in conjunction with control bars. The MDI client window is the direct parent of MDI child frame windows . the WS_HSCROLL and WS_VSCROLL window styles specified on a CMDIFramdWnd apply to the MDI client window rather than the main frame window so the fuser can scroll the MDI client area (as in the Windows Program Manager, for example).

The MDI frame window owns a default menu that is used as the menu bar when there is no active MDI child window. When there is a active MDI child, the MDI frame window's menu bar is automatically replaced by the MDI child window menu.

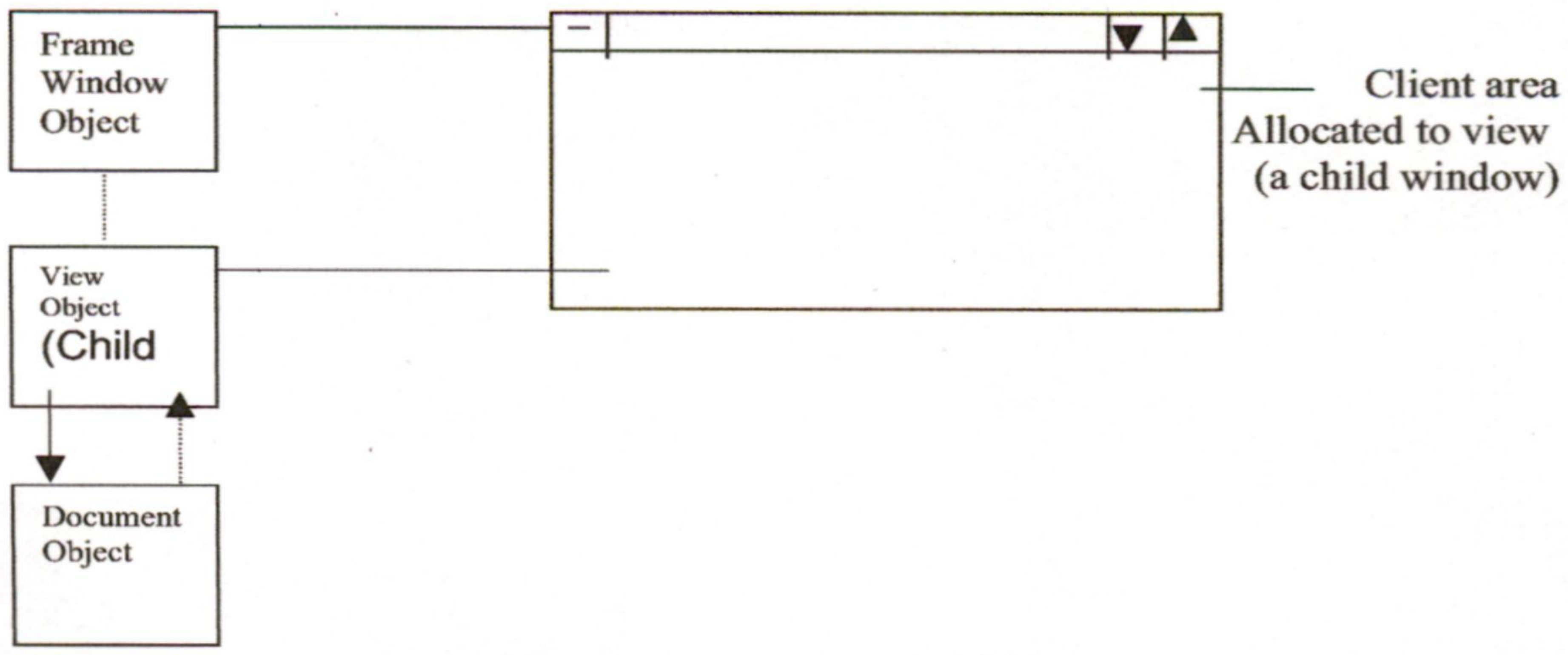
The MDI frame window works in conjunction with the current MDI child window, if there is one. For instance, command messages are delegated to the currently active MDI child before the MDI frame window.

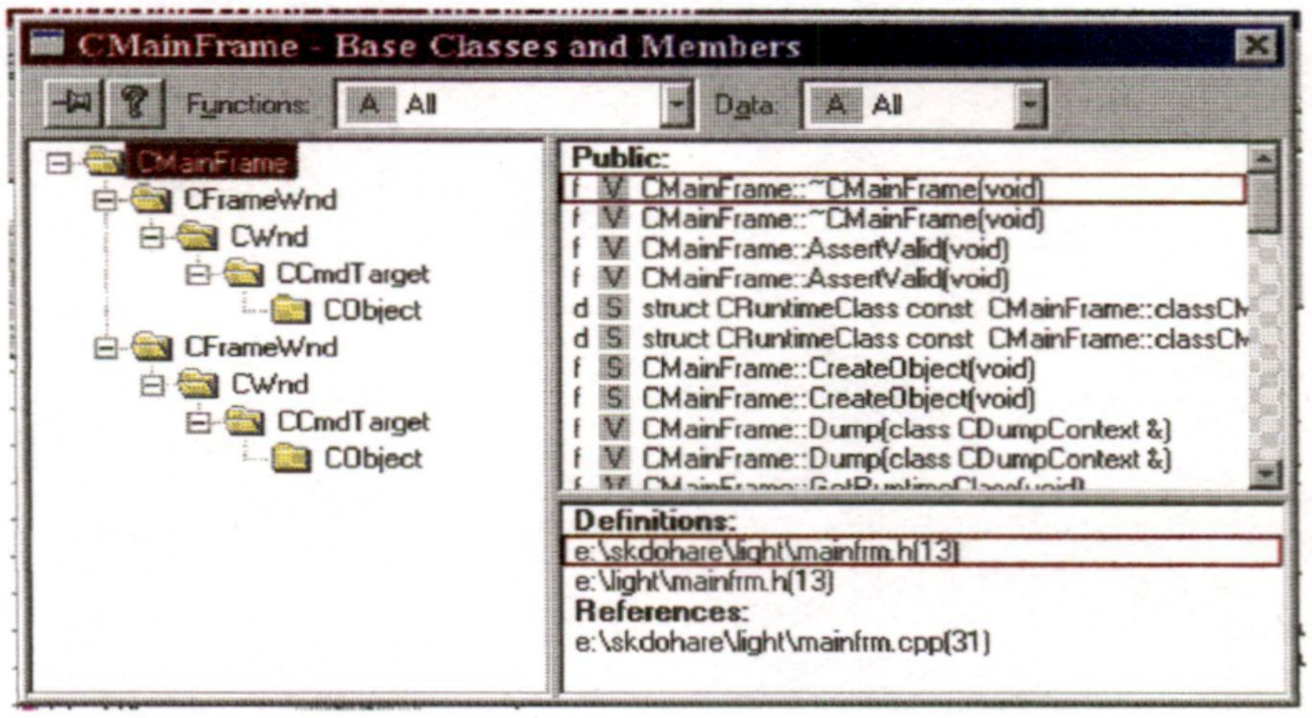
The MDI frame window has default handlers for the following standard Window menu commands:

ID_WINDOW_TILE_VERT
ID_WINDOW_TILE_HORZ
ID_WINDOW_CASCADE
ID_WINDOW_ARRANGE

The MDI frame window also has an implementation of ID_WINDOW_NEW, which creates a new frame and view on the current document . an application can override these default command implementations to customize MDI window handling.

Figure Frame Window and View





CMainFrame - Base Classes and Members

Functions: A All   Data: A All

```
⊟ CMainFrame
   ⊟ CFrameWnd
      ⊟ CWnd
         ⊟ CCmdTarget
            CObject
   ⊟ CFrameWnd
      ⊟ CWnd
         ⊟ CCmdTarget
            CObject
```

```
Public:
f V CMainFrame::~CMainFrame(void)
f V CMainFrame::~CMainFrame(void)
f V CMainFrame::AssertValid(void)
f V CMainFrame::AssertValid(void)
d S struct CRuntimeClass const  CMainFrame::classCM
d S struct CRuntimeClass const  CMainFrame::classCM
f S CMainFrame::CreateObject(void)
f S CMainFrame::CreateObject(void)
f V CMainFrame::Dump(class CDumpContext &)
f V CMainFrame::Dump(class CDumpContext &)
f V CMainFrame::GetRuntimeClass(void)
Definitions:
e:\skdohare\light\mainfrm.h(13)
e:\light\mainfrm.h(13)
References:
e:\skdohare\light\mainfrm.cpp(31)
```

Class Name: CNewScene

CNewScene is the Dialog Box which handles the complete scene design, scene storing onto .CUE files, individual scene testing, etc. for Light and Sound Control System.

The Base Class is CDialog and most of CDialog class members have been used.

Being the major Dialog Box it has several buttons associated for:

*Saving the Current .CUE file i.e. scenesettings

*GoTo a particular scene

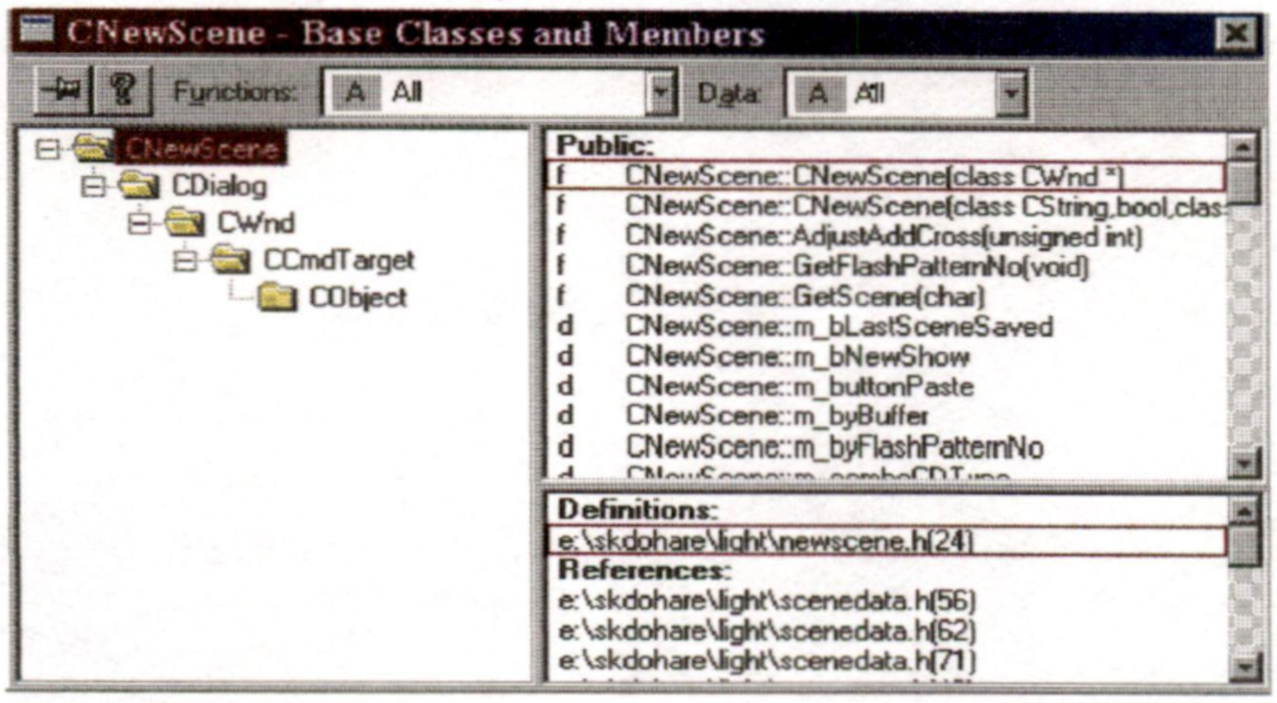
*Copy a particular scene onto current scene

*Firs, Next, Previous and Last Buttons for movement of scene numbers

*Test currently open scene

*Go to create cues(sequences)

*Ok and Back for user requests to exit with save or without save

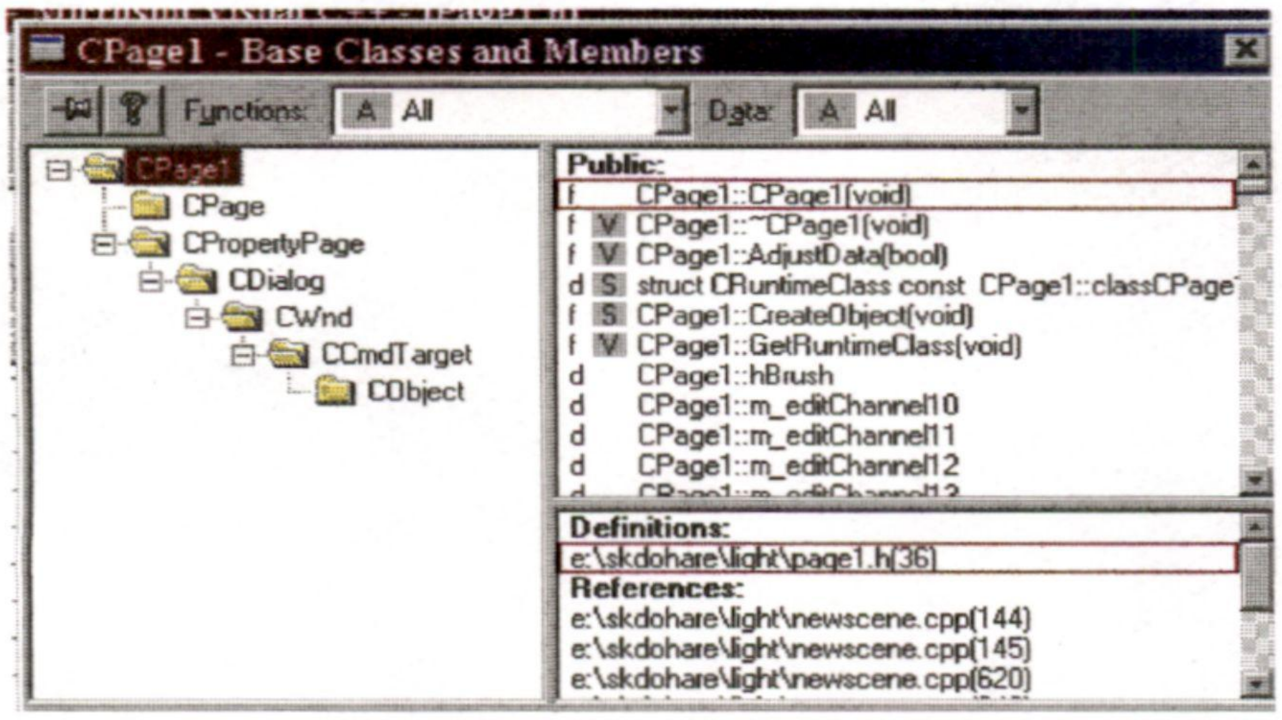


```
Public:
f     CNewScene::CNewScene(class CWnd *)
f     CNewScene::CNewScene(class CString,bool,clas
f     CNewScene::AdjustAddCross(unsigned int)
f     CNewScene::GetFlashPatternNo(void)
f     CNewScene::GetScene(char)
d     CNewScene::m_bLastSceneSaved
d     CNewScene::m_bNewShow
d     CNewScene::m_buttonPaste
d     CNewScene::m_byBuffer
d     CNewScene::m_byFlashPatternNo
```

```
Definitions:
e:\skdohare\light\newscene.h(24)
References:
e:\skdohare\light\scenedata.h(56)
e:\skdohare\light\scenedata.h(62)
e:\skdohare\light\scenedata.h(71)
```

Class Name: CPagen

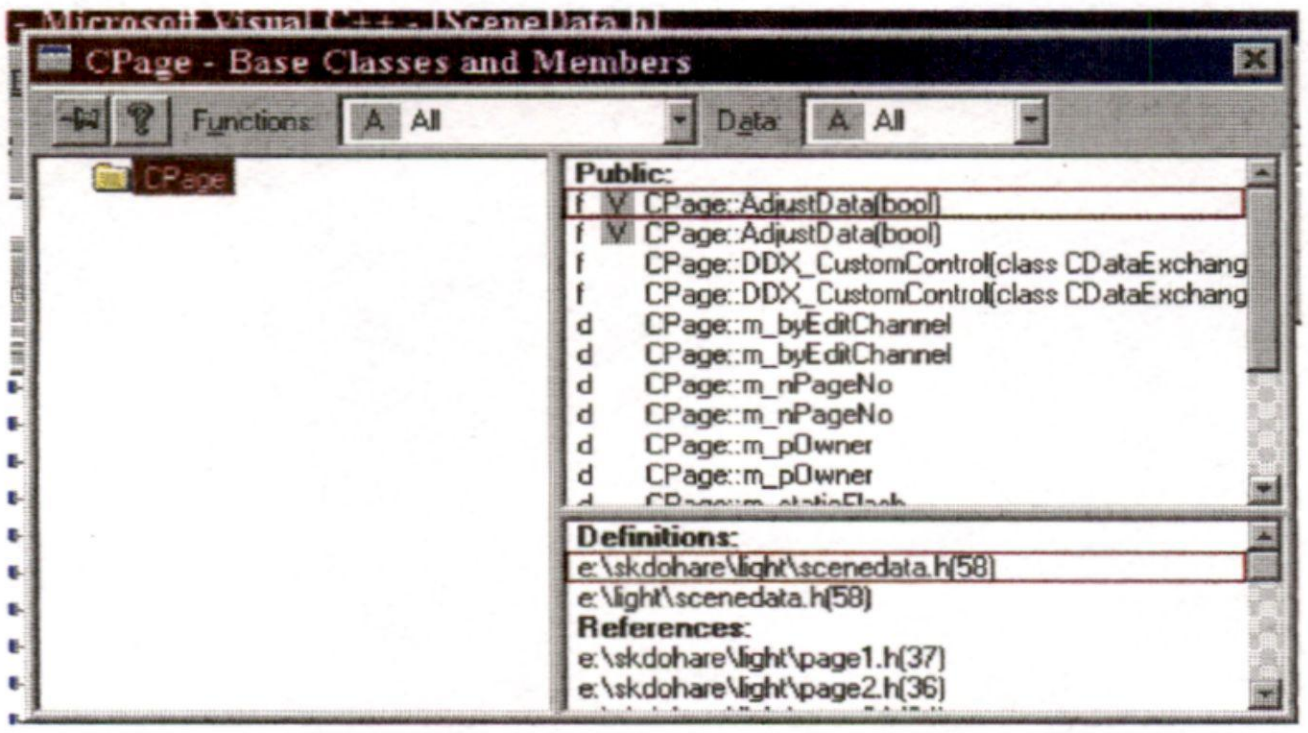
This class uses SetScrollRange and EnableScrollBar and SetScrollPos in InitDialog() function. It Scrolls the Virtical Scroll Bars and Sets the Maximum value 255 and Minimun value 0. it also enables the flash patterns and take input to all 32 channels.
OnKillfocusEditChannel17() kill the flash selection pattern



```
Public:
f      CPage1::CPage1(void)
f  V   CPage1::~CPage1(void)
f  V   CPage1::AdjustData(bool)
d  S   struct CRuntimeClass const  CPage1::classCPage
f  S   CPage1::CreateObject(void)
f  V   CPage1::GetRuntimeClass(void)
d      CPage1::hBrush
d      CPage1::m_editChannel10
d      CPage1::m_editChannel11
d      CPage1::m_editChannel12
Definitions:
e:\skdohare\light\page1.h(36)
References:
e:\skdohare\light\newscene.cpp(144)
e:\skdohare\light\newscene.cpp(145)
e:\skdohare\light\newscene.cpp(620)
```

Class Name: CPage

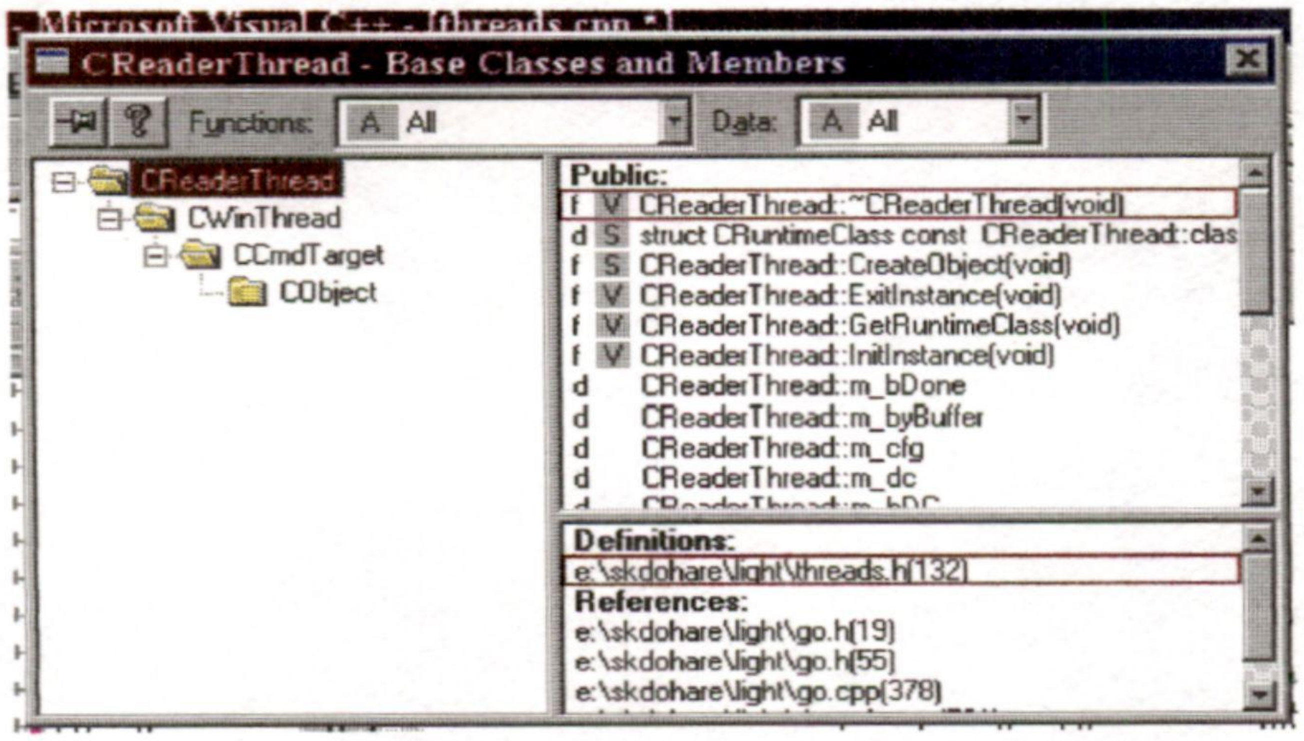
. This class is declare in **SceneData.h** header file and uses an Array of BOOL type **m_byEditChannel[32]** and **m_staticFlash[32]** and uses a **CNewScene** class object .



```
Public:
f  V   CPage::AdjustData(bool)
f  V   CPage::AdjustData(bool)
f      CPage::DDX_CustomControl(class CDataExchang
f      CPage::DDX_CustomControl(class CDataExchang
d      CPage::m_byEditChannel
d      CPage::m_byEditChannel
d      CPage::m_nPageNo
d      CPage::m_nPageNo
d      CPage::m_pOwner
d      CPage::m_pOwner
Definitions:
e:\skdohare\light\scenedata.h(58)
e:\light\scenedata.h(58)
References:
e:\skdohare\light\page1.h(37)
e:\skdohare\light\page2.h(36)
```

It also uses **AdjustData()** function and **SetOwnerShip** of Scene. It uses a PASCAL rutine **DDK_CustomContorl**

Class Name: CReaderThread


Microsoft Visual C++ - [threads.cpp *]
CReaderThread - Base Classes and Members
Functions: A All
Data: A All
CReaderThread
CWinThread
CCmdTarget
CObject

```
Public:
f V CReaderThread::~CReaderThread(void)
d S struct CRuntimeClass const CReaderThread::clas
f S CReaderThread::CreateObject(void)
f V CReaderThread::ExitInstance(void)
f V CReaderThread::GetRuntimeClass(void)
f V CReaderThread::InitInstance(void)
d   CReaderThread::m_bDone
d   CReaderThread::m_byBuffer
d   CReaderThread::m_cfg
d   CReaderThread::m_dc
Definitions:
e:\skdohare\light\threads.h(132)
References:
e:\skdohare\light\go.h(19)
e:\skdohare\light\go.h(55)
e:\skdohare\light\go.cpp(378)
```

This class declare in Thread.h header file and is a CWinThread type class. it uses a Run() function .

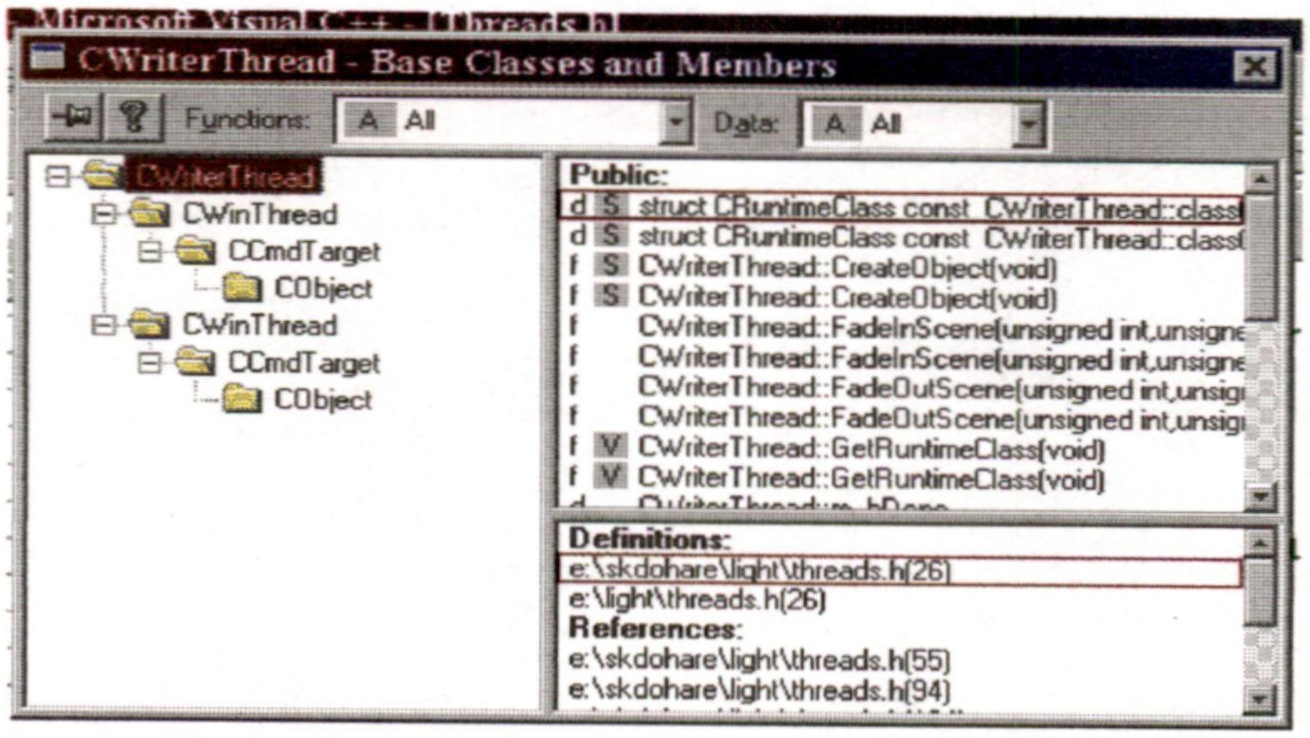
When **ReaderThread** is On pasition flash lock the **RingBuffer** so that no thread can modify the RingBuffer otherwise Flash will not work Properly.

ReadeThread reads values from **RingBuffer** and Put data into driver's buffer **TxPtr** from RingBuffer Before it , it uses bufferLock to lock the Ringbuffe .and send DMX 512 packet to specified serial com port or LPT1. this routines include MAB, SC, CD(up to 512) and function SendToPort();

When Open for ReaderThread is TURE the data is transferred from the driver's buffer to the DMX-Dongle by calling this function DongleReadWrite must only be called by the first user of this dongle. During this application another application is terminated. When data has been sent to screen it Unlock the ring buffer and switch off dimmers.

SendToScreen function draws the label and Dimmer block in black color.

Class Name: CWriterThread


Microsoft Visual C++ - [Threads.h]
CWriterThread - Base Classes and Members
Functions: A All
Data: A All

```
CWriterThread
  CWinThread
    CCmdTarget
      CObject
  CWinThread
    CCmdTarget
      CObject
```

```
Public:
d S struct CRuntimeClass const  CWriterThread::class
d S struct CRuntimeClass const  CWriterThread::class(
f S CWriterThread::CreateObject(void)
f S CWriterThread::CreateObject(void)
f   CWriterThread::FadeInScene(unsigned int,unsigne
f   CWriterThread::FadeInScene(unsigned int,unsigne
f   CWriterThread::FadeOutScene(unsigned int,unsig
f   CWriterThread::FadeOutScene(unsigned int,unsig
f V CWriterThread::GetRuntimeClass(void)
f V CWriterThread::GetRuntimeClass(void)
```

```
Definitions:
e:\skdohare\light\threads.h(26)
e:\light\threads.h(26)
References:
e:\skdohare\light\threads.h(55)
e:\skdohare\light\threads.h(94)
```

CWriterThread defines the CWinThread publicly and give the function SetOwner and FadeOutScene() and FadeInScene(). Which control the information about the Cross and add options and during running he show it Fade-out and Fade-in scenes.

CHAPTER: 5

TECHNICAL ISSUE

5.1 THE DMX512 PACKET

Before we go on to explain the structure of the DMX packet, I am assuming that we should have some knowledge of the bits & bytes that make up any computer's fundamental language system.

The DMX bits are also represented by a digital high (HI) or a digital low (LO).The actual DMX output transmits these HI and LO signals.

5.1.1 DMX512 PHYSICALS

The DMX data stream clocks out at the rate of 250Khz, which means each bit is measured at 4 microseconds widths.

1) IDLE or NO DMX situation: In the absence of a valid DMX packet the output of a DMX line will be a continuously HI signal.

2) BREAK : The start of a DMX packet is heralded by the output going LO for a MINIMUM period of 88 microsecs. This means 22 LO bits are measured out one after the other. This is known as the BREAK. The BREAK could be longer and up to 1 sec.

Experience shows that slightly longer breaks (above 88 microsecs) sent from a console are better since the receiving devices are generally given the algorithm={Is the BREAK>88 microsecs or 22 pulses}. I keep it generally at 120 microsecs in equipment designed by me

3) MARK AFTER BREAK (MAB) : The MAB immediately follows the BREAK by making the output go HI for a MINIMUM period of 8 microsecs or 2 pulses.

This MAB is a bit of a problem since the difference between the original DMX512 and the current DMX512 (1990) standards relate to this period of the packet. The original was set at 4 microsecs or 1 pulse. This created hassles for some receivers for being too short a MAB for detection and was upgraded to 8 microsecs or 2 pulses in 1990. The problem comes when a older console is used with newer receivers or vice versa. Wrong detection will lead to packet rejection or the wrong data going to the wrong channel. This will travel down the line leading to utter confusion. Some consoles and receivers have a dipswitch to set this parameter for both timings. Again the maximum MAB length could be 1 sec. My ideal timing would be 12 microsecs.

4) START CODE (SC) : The SC is next in the line. It is easier to remember that the SC is the start of the actual data stream where all individual channel data have the same format. The BREAK & the MAB were of different timings but the SC onwards all frames would have the same structure and timing of 11 pulses or 44 microsecs width. The first one can be termed, as data for channel No 0, which is a non-existent channel and represents the SC.

I will first describe the general structure of these channel data frames:

-Of the 11 pulses the 1st one is always LO signifying the Start bit.

-This is followed by the actual data byte of 8 bits (which could be any of 256 values from 0 to 255).

-The frame ends with 2 bits which are HI signifying the two stop bits and end of the channel information.

Channel No 0 is the SC, which as things stand, ALWAYS has the data byte = 0 signifying that the following data is for dimmers. As per the current standard, no other value can be used. The option is left open and as and when ESTA specifies, the SC value may be used to tell the receiver that the data following it is meant for a specific type of receiver. That is the end purpose of having the SC..... to be able to segregate a packet of data, receiver wise. But, for the moment, it's zero which has been specified for dimmers by ESTA. Do remember that this also includes just about any receiving device like dimmers, scans or whatever!

5) MARK TIME BETWEEN FRAMES (MTBF) : The mark time between frames can be from 0 sec up to 1 sec but the lesser the better. Each channel frame can have the MTBF before the start bit. The MTBF is obviously HI.

6) CHANNEL DATA (CD) : The CD frames follows the SC frame in a logical manner from 1 to 512 (or less) in the form described above.

7) MARK TIME BETWEEN PACKETS (MTBP) : After the last valid CD stop bits are sent, one full packet is completed and the next packet can start with a fresh BREAK & MAB. However an idle (HI) can be inserted between packets (MTBP), the length of which may be 0 sec to 1 sec. It is upto the console designer to design the architecture of the console such that the thru-put time is at a minimum.

9.1.2 IMPORTANT

The wonderful part of DMX is that you DONT send the CHANNEL NUMBER AT ALL!!
The 1st byte AFTER the STARTCODE (which is always 0) is AUTOMATICALLY taken as the data value for Channel 1...next, data value for Channel 2....next, data value for Channel 3...and so on...upto 512 OR less channels. That is how the receivers...be they intelligent lights or whatever...will decode them. Actually a channel counter is set up in the receiver...either in the microprocessor it self or as a hardware counter. This counter will be automatically reset to point at channel 0 when a valid BREAK and a valid MAB is detected. Subsequently with every LAST stop bit in each frame, this counter is incremented by ONE. Thus, DURING the SC frame, the counter output reads zero. At the end of the SC (last stop bit of the SC frame)the counter output

becomes one, thus telling the processor that the next frame contains the data for channel 1 and so on. So the receiver knows which channel the current data is valid for.... Thus when you set a start address in an MARTIN 812 at say 50 (+6 channels) it simply takes the 6 data bytes from when the internal channel counter reaches 50.... counting upto 55!! The moment you start a fresh BREAK ...etc sequence (a new packet, that is) this counter is reset!! So it is fully legal for a console or software to generate upto valid 100 data bytes after the SC for 100 channels and then generate a BREAK, etc. Thus you don't HAVE TO generate all 512 bytes !! The LSC ATOM, for instance, has the capacity to drive 99 channels; thus at the end of the 100th frame or a count of 99(remember, we have an SC frame to count, too, at the count of zero) the console starts sending a BREAK + MAB and so on. This concept is vital in order to understand the one to one relation between channel nos and their respective data.

The following is a mathematical layout of the typical DMX512 (1990) timing: -

[(88)+(12)+(44)+(CHL*44)+(CHL*MTBF)+(MTBP)] microsecs

Where CHL is the no of Channels under consideration.

The ideal timing (this is purely my personal opinion) :-

[(120)+(12)+(44)+(CHL*44)+(0)+(50)] microsecs

For 512 channels my timing would be 22754 microsecs.

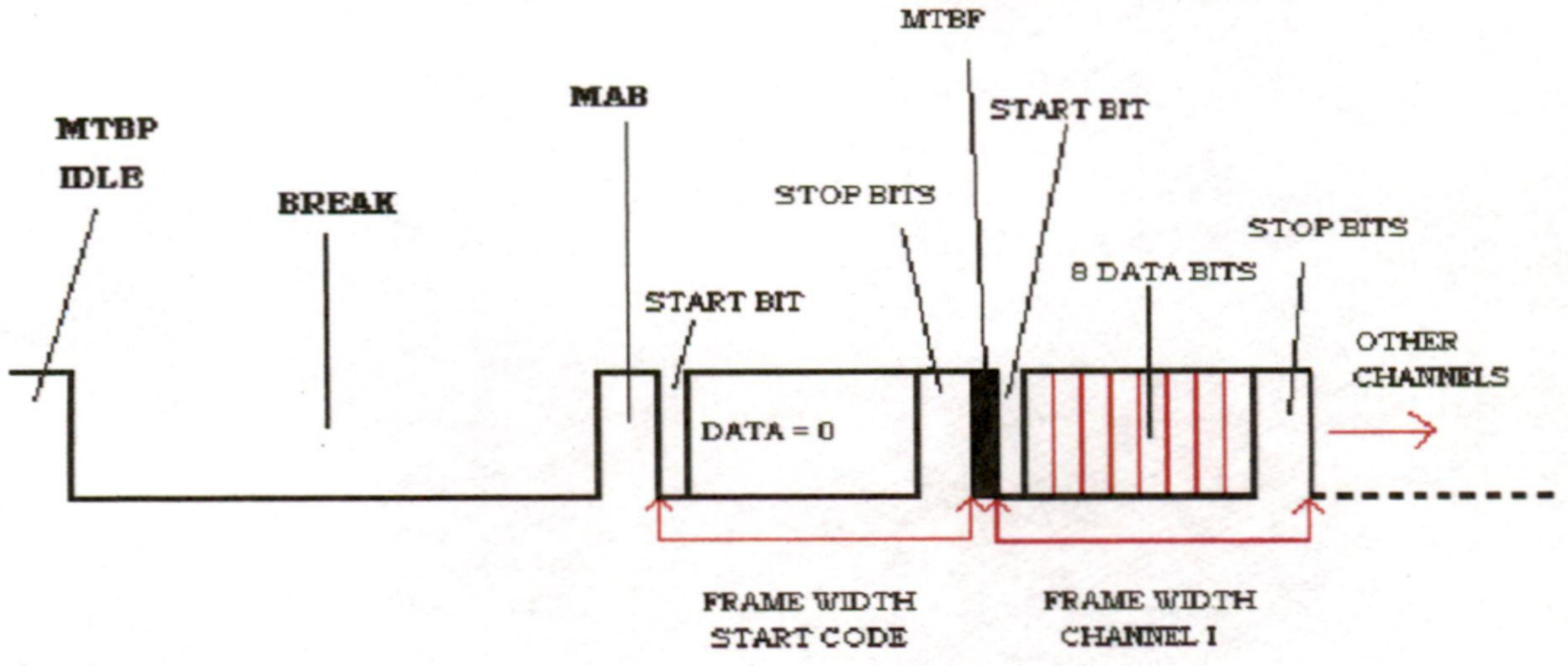
Thus the refresh rate = 1000000 / 22754 = 43.9 or 44 Hz

How the refresh rate varies would depend upon the Microprocessor speed, firmware code and system architecture.

DMX512 (1990) timing chart

Description	MIN	TYP	MAX	UNIT
BREAK	88	88	1000000	usec
MAB		8		usec
FRAME WIDTH		44		usec
START/DATA/STOP BITS		4		usec
MTBF	0	NS	1000000	usec
MTBP	0	NS	1000000	usec

Note: NS means Not Specified and is designer definable



DMX512 TIMING DIAGRAM

5.2 DMX-Dongle II WINDOWS 95 98 NT

This contains all the information needed to develop 32 bit applications for Windows 95, Windows 98 and Windows NT Ver 4.0.

5.2.1 Hardware Features :

The DMX-Dongle II provides the following features:

=> DMX512 transmission of 512 channel data.
=> Programmable transmit Break, Mark after Break and Header Code
=> DMX512 receive with programmable base address
=> Receive 512 channels @ 8 bit resolution and full bandwidth
=> Simultaneous transmit and receive
=> Programmable Merge and Loop Through
=> Programmable receive Header Code
=> Analysis of receive errors
=> Receive Break and Mark after Break timing analysis
    ⇨ Multiple oscilloscope trigger outputs

5.2.2 Software Features :

The DMX-Dongle II Windows SDK provides a common development platform which supports the three main Microsoft operating systems.

Key new features include:

=> Application access to a total of three DMX-Dongle II
=> Multiple application access to a single DMX-Dongle
=> Stealth Mode allowing lighting control and lighting visualization software to share data
=> Continuous transmit and receive without PC intervention
=> Receive of entire frame without PC intervention
=> Programmable Merge from input or Loop Through

5.3DIRECTORY STRUCTURE

The SDK is installed into the path: C:\Program Files\DMX-DongleII\SDK.

The folder structure is as follows:

=> Bin Contains all the device driver files.

à Wrtdev*.vxd Windows 95 / 98 drivers.

Target: C:\Windows\System\Vmm32

à WinRT.sys Windows NT Driver.

Target: C:\WinNT\System32\Drivers

à DongNT.Dll Main Driver.

Target (95/8): C:\Windows\System

Target (NT): C:\WinNT\System32

=> DongleView Application and source files for DongleView.exe. This is a C++Builder Ver3 project.

à DongleView.bpr Borland project file

à DongleView.exe Application

à DongleView.cpp Project source code

à Unit1.cpp Application source code

=> Include The C++ include files required to access the DLL.

à DongNT.hpp Include file for driver. Includes all DLL call function prototypes and global defines.

à DongleDllLoad.cpp Example code to dynamically link to DLL.

à DongleDllLoad.hpp Include file for above.

=> InstallShield Contains an example InstallShield project which contains all the required settings to install the DMX-Dongle Drivers and update the registry.

à DongleIISample.iwz InstallShield project file. Use this as a
template for deploying DMX-Dongle applications.

5.4 S D K

=> Registry Contains details of the registry changes executed when installing DMX-Dongle software.

à DongleReg.reg Regedit command file to install Registry keys required for the DMX-Dongle Drivers.

All of the files contained in the SDK may be redistributed subject to the licence agreement contained in the SDK folder. Where the files detailed in the BIN folder are redistributed, they must be installed in the target folders specified above and must not be renamed.

5.4.1 DLL Access : All access to the DMX-Dongle II is routed via the DongNT Dll. The Dll operates under Windows 95, Windows 98 and Windows NT Ver4, allowing multiple applications access to a maximum of three DMXDongles.

The windows operating systems allow access to a DLL via two methods, static or dynamic linking. However, the DongNT Dll will only operate correctly when accessed dynamically. Example code showing how to dynamically link the functions is included in the SDK\Include folder.

5.4.2 Initialising the Dongle : The DMX-Dongle is initialised with the command:

Type=DongleInit(Port,Aux,MaxChannel);

Port specifies one of the three DMX-Dongles that are connected to LPT1,2 or 3. Aux is set to zero and MaxChannel is set to a number between 1 and 512. It represents the number of DMX channels that will be processed by the driver. The function returns an unsigned int which is broken into two bytes. The high byte is the status return from the driver, whilst the low byte is the status return from the DMX-Dongle. The return codes are detailed in DongNT.hpp:

```
#define DiLoGood 0x0005
#define DiLoSystemFail 0x0045
#define DiHiGood 0x0000
#define DiHiBadDriver 0x0100
```

```
#define DiHiAlreadyOpen 0x0200
#define DiHiBadPortNumber 0x0300
```

The following to pseudo code examples show a simple and a sophisticated approach to initialising. Simple Initialisation:

```
unsigned char Open=0; // Set when a dongle is open
unsigned char PrimaryUser=0; // Set if dongle first opened by this app
unsigned char Port=0; // 0-2 current dongle port
unsigned int Type=0;
unsigned char* RxPtr=0;
unsigned char* TxPtr=0;
```

5.5 FUNCTION PROTOTYPES

DongleClose

Purpose: Close an application's attachment to a DMX-Dongle.

Syntax: unsigned char DongleInit(unsigned char Port)

Remarks: The function will fail harmlessly if DongleInit was unsuccessful.

Arguments: Port: 0-3 represents Lpt1-Lpt3

Return Value: True if the close action was successful

DongleGet

Platform Purpose: Detect the operating system.

Syntax: unsigned char DongleGetPlatform(Void)

Remarks:

Arguments: None

Return Value: True if NT or False if 95/98 or Win 3.1 running Win32.

DongleGet

InUse Purpose: Return the number of applications attached to this DMXDongle.

Syntax: unsigned char DllDongleGetInUse(unsigned char Port)

Remarks:

Arguments: Port: 0-2

Return Value: Zero indicates no applications attached. A maximum of 255 applications may theoretically attach to a DMXDongle.

DongleGet

TxPtr Purpose: Return a pointer to the array of 512 unsigned char which represents the transmit level buffer.

Syntax: unsigned char* DongleGetTxPtr(unsigned char Port)

Remarks: The buffer is output to the DMX-Dongle by each call to DongleReadWrite

Arguments: Port: 0-2

Return Value: unsigned char *

DongleGet

RxPtr Purpose: Return a pointer to the array of 512 unsigned char which represents the receive level buffer.

Syntax: unsigned char* DongleGetRxPtr(unsigned char Port)

Remarks: The buffer is updated by the DMX-Dongle on each call to DongleReadWrite

Arguments: Port: 0-2

Return Value: unsigned char *.

DongleGet

Stealth Purpose: Return the Stealth Mode of the DMX-Dongle.

Syntax: unsigned char DongleGetStealth(unsigned char Port)

Remarks: When Stealth Mode is active, the physical DMX512 input to the DMX-Dongle is ignored, instead the transmit data is automatically copied to the receive buffer. The purpose of this mode is to allow control and visualisation packages to coexist on the same PC.

Arguments: Port: 0-2

Return Value: True if Stealth Mode active

DongleSet

Stealth Purpose: Enable or disable the Stealth Mode of the DMX-Dongle.

Syntax: unsigned char DongleSetStealth(unsigned char Port, unsigned char Stealth)

Remarks: When Stealth Mode is active, the physical DMX512 input to the DMX-Dongle is ignored, instead the transmit data is automatically copied to the receive buffer. The purpose of this mode is to allow control and visualisation packages to coexist on the same PC. Stealth mode should only be enabled at the request of the user. Stealth mode should not be enabled if only one user is attached to the DMX-Dongle.

Arguments: Port: 0-2

Stealth: True enables Stealth Mode.

Return Value: True if Stealth Mode active

DongleGet

Max Purpose: Return the number of channels that are currently refreshed by the driver.

Syntax: unsigned int DongleGetMax(unsigned char Port)

Remarks:

Arguments: Port: 0-2

Stealth: True enables Stealth Mode.

Return Value: The current number of channels processed by the driver (1-512).

DongleSet

Max Purpose: Set the number of channels that are currently refreshed by the driver.

Syntax: unsigned int DongleSetMax(unsigned char Port, unsigned int Max)

Remarks: The DMX-Dongle always transmits and receives all 512 channels. Significant processing power improvements are obtained by reducing the number of channels that are processed by the driver. In a control application this value will typically be set on the fly as lamps are patched. When multiple applications are attached to a DMX-Dongle, this function only allows the Max Value to be increased.

Arguments: Port: 0-2

Max: 1-512

Return Value: The current number of channels processed by the driver (1-512).

DongleRead

Write Purpose: Transfer all data and configuration between driver buffers and DMX-Dongle hardware.

Syntax: unsigned int DongleReadWrite(unsigned char Port)

Remarks: This function must only and always be called continuously by the first application to attach to a DMX-Dongle.

Arguments: Port: 0-2

Return Value: Dependent upon hardware.

DongleReset

RxCounters Purpose: Reset to zero the DMX-Dongle's internal receive data and error counters.

Syntax: void DongleResetRxCounters(unsigned char Port)

Remarks:

Arguments: Port: 0-2

Return Value:

DongleClear

DataBuffer Purpose: Zeroes the transmit and receive buffers.

Syntax: void DongleClearDataBuffer(unsigned char Port, unsigned char Data)

Remarks: Sets both the transmit and receive buffers to a common value which is usually zero.

Arguments: Port: 0-2

Data: 0-255, the data value used to fill the buffers

Return Value: None

DongleSetRx

Header Purpose: Set the Start Code deemed valid for received DMX512. This value is usually zero.

Syntax: void DongleSetRxHeader(unsigned char Port, unsigned char Head)

Remarks: Any frames received which do not match this start code are logged by the DongleGetHeaderErrorCount function. These frames are also not copied to the receive buffer.

Arguments: Port: 0-2

Head: Receive Start Code 0-255

Return Value: None

DongleSetTx

Header Purpose: Set the Start Code used to transmit all DMX512 frames, This value is usually zero.

Syntax: void DongleSetTxHeader(unsigned char Port, unsigned char Head)

Remarks: Ait is usual to use a zero value for the Start code.

Arguments: Port: 0-2

Head: Transmit Start Code 0-255

Return Value: None

DongleSetTx

BreakTime Purpose: Set the break time in micro-seconds used for all DMX512 transmission.

Syntax: void DongleSetTxBreakTime(unsigned char Port, unsigned char Time)

Remarks: This value must not be set to less than 88uS for normal operation. A value of 200uS is suggested. When controlling the DMX-Dongle I this is an absolute value. When controlling the DMX-Dongle II, this is a minimum value which can be expected to fluctuate.

Arguments: Port: 0-2

Time: 0-255 uS break time.

Return Value: None

DongleSetTx

MabTime Purpose: Set the mark after break time in micro-seconds used for all DMX512 transmission.

Syntax: void DongleSetTxMabTime(unsigned char Port, unsigned char Time)

Remarks: This value must not be set to less than 8uS for normal operation. A value of 20uS is suggested. When controlling the DMX-Dongle I this is an absolute value. When controlling the

DMX-Dongle II, this is a minimum value which can be expected to fluctuate.

Arguments: Port: 0-2

Time: 0-255 uS Mab time.

Return Value: None

DongleGet

Firmware Revision

Purpose: Return the DMX-Dongle firmware revision.

Syntax: unsigned char DongleGetFirmwareRevision (unsigned char Port)

Remarks: Three versions exist:

0x16 = Pre-production DMX-Dongle I

0x17 = Production DMX-Dongle I

0x20 = Production DMX-Dongle II

The DLL is capable of driving all versions, however the following issues must be considered:

The DMX-Dongle I requires PC intervention to transmit or receive an entire frame, whereas the DMX-Dongle II autonomously transmits and receives.

The DMX-Dongle I is only capable of half duplex operation, whereas the DMX-Dongle II can transmit and receive simultaneously.

The DMX-Dongle I does not have loop or merge facilities and also communicates with the PC at approximately 20% of the rate of the DMX-Dongle II.

Arguments: Port: 0-2

Return Value: Firmware Revision

DongleGet

Type Purpose: Return the type code for the DMX-Dongle.

Syntax: unsigned int DongleGetType(unsigned char Port)

Remarks: This function provides the same return as DongleInit.

Arguments: Port: 0-2

Return Value: Type

DongleSet

TxMode Purpose: Enable the DMX512 output.

Syntax: void DongleSetTxMode(unsigned char Port)

Remarks: This function operates slightly differently depending upon the DMX-Dongle firmware:

Firmware = 0x16 or 0x17 (DMX-Dongle I): Switches the DMX-Dongle to transmit mode and therefore disables received DMX512. This function should be called whenever it is necessary to switch to transmit mode.

Firmware = 0x20 (DMX-Dongle II): Enables DMX512
transmission, does not affect received DMX512. This function should be called once at initialisation if the transmit capability is required.

Arguments: Port: 0-2

Return Value:

DongleSet

RxMode Purpose: Enable the DMX512 input.

Syntax: void DongleSetRxMode(unsigned char Port)

Remarks: This function operates slightly differently depending upon the DMX-Dongle firmware:

Firmware = 0x16 or 0x17 (DMX-Dongle I): Switches the DMX-Dongle to receive mode and therefore disables DMX512 transmit. This function should be called whenever it is necessary to switch to receive mode.

Firmware = 0x20 (DMX-Dongle II): No action

Arguments: Port: 0-2

Return Value:

DongleGet

Channel

Count

Purpose: Return the number of channels contained in the last received DMX512 frame.

Syntax: unsigned int DongleGetChannel Count(unsigned char Port)

Remarks: Use DongleResetRxCounters to clear the count.

Arguments: Port: 0-2

Return Value: 1-512

DongleGet

BreakTime Purpose: Return the break time in microseconds of the last received DMX512 frame.

Syntax: unsigned int DongleGetBreakTime(unsigned char Port)

Remarks: Values lower than 88uS are outside the DMX512 specification.

Arguments: Port: 0-2

Return Value: Break time

DongleGet

MabTime Purpose: Return the mark after break time in microseconds of the last received DMX512 frame.

Syntax: unsigned int DongleGetMabTime(unsigned char Port)

Remarks: Values lower than 8uS are outside the DMX512 (1990) specification.

Arguments: Port: 0-2

Return Value: Mab time

DongleGet

Period Purpose: Return the total frame time in milliseconds of the last received DMX512 frame.

Syntax: unsigned int DongleGetPeriod(unsigned char Port)

Remarks: The reciprocal of this value gives the data refresh rate.

Arguments: Port: 0-2

Return Value: Frame period

DongleGet

Overrun

ErrorCount

Purpose: Return the total number of overrun errors detected in then received DMX512 data stream.

Syntax: unsigned int DongleGetOverrunError Count(unsigned char Port)

Remarks: Use DongleResetRxCounters to clear the count.

Arguments: Port: 0-2

Return Value: Error count

DongleGet

Framing

ErrorCount

Purpose: Return the total number of framing errors detected in the received DMX512 data stream.

Syntax: unsigned int DongleGetFramingError Count(unsigned char Port)

Remarks: Use DongleResetRxCounters to clear the count.

Arguments: Port: 0-2

Return Value: Error count

DongleGet

Header

ErrorCount

Purpose: Return the total number of DMX512 frames received with non matching start codes.

Syntax: unsigned int DongleGetHeaderError Count(unsigned char Port)

Remarks: Use DongleResetRxCounters to clear the count.

Arguments: Port: 0-2

Return Value: Error count

DongleGet

FrameCount Purpose: Return the total number of DMX512 frames received.

Syntax: unsigned int DongleGetFrame Count(unsigned char Port)

Remarks: Use DongleResetRxCounters to clear the count.

Arguments: Port: 0-2

Return Value: Frame count

DongleGet

DllRevision Purpose: Return the revision number of the Dll.

Syntax: unsigned char DongleGetDllRevision (unsigned char Port)

Remarks:

Arguments: Port: 0-2

Return Value: Dll revision

DongleGet

BlockSize Purpose: Return the number of channels processed by each DongleReadWrite function call.

Syntax: unsigned int DongleGetBlockSize(unsigned char Port)

Remarks: The return depends upon the firmware revision. When a DMX-Dongle II is in use, each call to DongleReadWrite transfers an entire frame of both receive and transmit data between the driver and the Dongle.

Arguments: Port: 0-2

Return Value: Firmware = 0x16, Return = 128

Firmware = 0x17, Return = 256

Firmware = 0x20, Return = 512

DongleSet

Output Purpose: Set the DMX512 output mode.

Syntax: unsigned char DongleSetOutput(unsigned char Port, unsigned char Output)

Remarks: Normal mode is defined as transmitted DMX512 comprises of the transmit buffer. Loop mode retransmits received DMX512 to the output. Merge mode combined received DMX512 and the transmit buffer on a highest takes precedence basis.

Arguments: Port: 0-2

Output: 0 = Normal

1 = Loop

2 = Merge

Return Value: True if Firmware = 0x20 (DMX-Dongle II)

DongleGet

Output Purpose: Returns the current DMX512 output mode.

Syntax: unsigned char DongleGetOutput(unsigned char Port)

Remarks: Normal mode is defined as transmitted DMX512 comprises of the transmit buffer. Loop mode retransmits received DMX512 to the output. Merge mode combined received DMX512 and the transmit buffer on a highest takes precedence basis.

Arguments: Port: 0-2

Return Value: 0 = Normal

1 = Loop

2 = Merge

DEVELOPMENT ENVIRONMENT

VISUAL C++ 6.0 USING MFC

WIN 32 SDK

VISUAL C++ VER 6.0

USING

MICROSOFT FOUNDATION

CLASSES

CHAPTER: 6

DEVELOPMENT ENVIRONMENT

6.1 MICROSOFT VISUAL C++ VERSION 6.0.....

The Microsoft visual C++ version 6.0 development system for Windows 98 / Windows NT is an integrated development environment for C and C++ applications, with support multi-platform and cross-platform development. It includes a C++ application framework, the Microsoft foundation class library version 6.0, which facilitates the development of applications for Windows as well as the parting of application to multiple platforms. We can easily develop an application for windows on one platform using visual C++ and Microsoft foundation class library (MFC), and then use the same code to build application for other platforms.

The visual C++ development environment contains the visual elements that are used to manage our development efforts. It provides App wizard and custom aPP wizard for creating C++ projects based on the Microsoft foundation class library (MFC). The projects contain skeleton files that create an application framework with basic window functionality. We can than add our application –specific code to these files. In regular and logical ways using App wizard reduces our development time and effort by creating the initial skeleton files custom App wizard allows we initially create to our particular needs. This permits us to reuse code that we always include in our applications, thus, further reducing our development time.

The development environment also includes component gallery with contains reusable components that we can add to existing applications A component can be an OLE control our own reusable C++ classes with any associated resources or a component created by a third – party vendor. Components created by vendors can range from reusable code to full – fledged tools such as a code analysis tool.

As we develop our applications we generally need to revise them to eliminate bugs in our code. The development environment provides a number of capabilities to aid us in this debugging process, it not only helps us find the bugs but can provide assistance in identifying their causes.

6.1.1 COMPONENT GALLERY:

Software reuse is a key theme in visual C++, and component gallery is a great way to reuse software components.

Component gallery contains reusable code such as OLE controls, our own reusable C++ classes with any associated resources, or components created by third party vendor. Third party - created range components range from reusable code to useful tools, such as a code analysis tool.

Components gallery makes it easy to reuse a new dialog box or a control without cutting and pasting and without danger of name collisions.

We can use components from component gallery in our application, and we can add components to component gallery for later reuse. We can also share them with others.

We use App wizard to create a Windows based application that uses the MFC library, App wizard generates a complete feature-rich set of starter files from which we can build an application, whether that application is based on a single- document, multiple document, or dialog architecture, requires OLE or ODBC support targets multiple platforms, or generates an executable file or DLL.

But what about those specials types of application that is unique to our work? What do we do if our clients or we need applications with features that the standard App wizard doesn't handle? The answer is we create custom App wizards.

Custom App wizard is useful for creating generic application types that can be used over and over again. Custom App wizards are not useful for creating one-off application.

6.1.2 How App wizard works with MFC:

App wizard uses the Microsoft foundation class library (MFC) to help us build application for Windows. We can write MFC application without App wizard; but using App wizard, however, greatly simplifies and speeds our work.

For each project, App wizard creates an application class derived from the MFC CwinApp class. App wizard also generates an implementation file with the following features:

A message map for the application class.

An empty class constructor.

A variable that declares an object of the application class.

A standard implementation of the application class's InitInstance member function.
The application class is placed in the project header and main source files. The names of the class and files created are based on the project name we supply in App wizard.
We can use App wizard to create applications that use the following types of interfaces:

Interface	description
Single document (SDI)	An application that will open only one Document at a time – similar to notepad.
Multiple document(MDI)	An application that will open multiple documents At a time – similar to Microsoft word.
Dialog based	An application that uses a dialog box as an Interface.

App wizard allows us to generate projects that will create applications to run on the following platform.

Platform name	Comes with
Win 32(x86)	Visual C++
Win 32(MIPS)	Visual C++ RISK edition
Win 32(ALPHA)	Visual C++ RISK edition
Win 32(Power PC)	Visual C++ RISK edition
Macintosh	Visual C++ cross development edition for Macintosh
power Macintosh	Visual C++ cross development edition for Macintosh

App wizard and its language DLLs. those with names described by Appzx*.DLL- simplifythe process of localizing applications.

Our application can choose from the following features:
OLE automation and OLE container ,server, and miniserver support.
Database(ODBC or DAO) support.
MAPI support that enables us to write mail-aware applications.
Windows sockets support that allows us to write TCP/IP application.

Our application depending on whether it is SDI, MDI, or dialog based, can choose from the following window features:
Dockable toolbar

Initial status bar
Printing and print preview
Context sensitive help
3D Control

MFC Hierarchy chart: In order to use MFC , we must understand the relationship between a class and its base class, and between a class and its derived classes. the MFC Hierarchy Chart provides a quick, visual interpretation of the inheritance relationship of the classes in the MFC.

The Microsoft foundation class library provides collection classes to manage groups of objects.

Collection classes:

The collection classes are of two types:
Collection classes created from C++ templates
Collection classes not created from templates

A collection class is characterized by its "shape" and by type of its elements. MFC provides collections in three shapes:

Lists:

The list class provides an ordered, non-indexed list of elements, implemented as doubly linked list. A list has a "head" and a "tail" and adding or removing elements from the head or tails, or inserting or deleting elements in the middle, is very fast.

Arrays:

The array class provides a dynamically sized, ordered, and integer-indexed array of objects.

Maps:

A map is collection that associates a key object with a value object.

OLE Control:

An OLE control is a reusable software component that supports a wide variety of OLE functionality and can be customized to fit many software needs. These controls can be developed for variety of uses, including database access, data monitoring, and graphing.

Besides their portability, OLE controls support features previously not available to custom controls, such as compatibility with existing OLE containers and the ability to integrate their menus with OLE container menus. In addition, an OLE control fully supports OLE automation, which allows the control to expose writable properties and set of functions that can be called by the control's user.

MFC provides a set of classes that we can use to implement OLE controls. In addition to the OLE control classes, OLE control macros and globals implement, among other things, control serialization and property exchange. The following is a list of the OLE control classes:

CConnection Point	Implements outgoing interfaces.
CFont Holder	Implement font properties.
Cpicture Holder	Implements picture properties.
C OLE Control	Implement the OLE control framework.
C OLE Control module	Implements an OLE control module.
C OLE Property page	Implements an OLE control property page.
C Prop Exchange	Implements the exchange of OLE control properties.

OLE control containers are OLE clients that are capable of loading and supporting OLE controls. Any existing application, which can work as an OLE client, can insert an OLE control. However, because OLE controls provide a new architecture, they can not completely support the full functionality inherent in OLE controls.
For this reason, test container, shipped as a part of visual C++, provide support for complete testing of an OLE control.

6.1.3 Development Platform Which Are Containers For OLE Controls:

Currently, FoxPro 3.0, Access 2.0, Visual basic 4.0 and OLE containers built with Visual C++ version 6.0 fully support OLE controls. This container type, usually referred to as a control container, is able to operate our OLE control by using the control's properties and methods, and receive notifications from our OLE control in the form of events.

Converting a VBX Control to an OLE Control:

If we wish to use a VBX control as a template for a new OLE control, we must have the source code for the VBX control to do any real conversion work. This means that we can not convert a third-party VBX control to an OLE control without the source for the control – we will be dependent on the supplier of the control for a new version that use OLE control architecture.

Warning the VBX template tool was part of the control wizard found in version 1.0 and 1.1of CDK.This tool was removed from the OLE and properties.

OLE is an extensible technology for interapplication communication with a number of sub technologies, as described in the list below. Initially, OLE supported creating compound document, documents that could contain objects created by other application, both embedded and linked in document. In fact, this is the origin of the name OLE, "object linking and embedding" compound documents are still an important part of OLE technology. In fact, one of the newer features supported by OLE are especially for OLE documents, in-place activation or visual editing.

However, OLE is now much more than that. OLE provides data transfer in windows, not only drag-and-drop but also the clipboard and the underpinning for DDE.

There are two other major technologies supported by OLE and MFC, OLE Automation and OLE controls. Automation lets one application control another. It can support a common scripting language across multiple application from different vendors. OLE controls are reusable software components.

TOPIC	CLASS
OLE Documents	C OleclientItem, C OleserverItem, C OleDocument
In-place Activation	C OleIpframeWnd, C OleResizebar
Data transfer	C Oledatasource, C Oledataobject
Drag-and-drop	C Oledropsource, C Oledroptarget, Cview
Automation	C Oledispatch driver, C cmd target
OLE Controls	C Ole control, C Connection point

Key Components of an MFC application:

~~An application object, which represents our~~ application

Document template objects, which create document, frame window, and view objects.

Document objects, which store data and serialize it to persistent storage.

View objects, which display a document's data and manage the user's interaction with the data.
Frame window objects, which contain views.
Thread objects, which let us, program multiple threads of execution using MFC classes.
Dialog boxes, controls, and control bars, such as toolbars and status bars.
OLE visual editing and OLE automation.
OLE controls and the classes and tools are used to develop them.
Database support using open database connectivity (ODBC) and data access objects (DAO).
Useful general-purpose classes, such as strings collections, exceptions, and date/time objects.

Taken together, the classes in the Microsoft foundation class library (MFC) make up an "application framework" the framework on which we built an application for windows. At a very general level, the framework defines the skeleton of an application and supplies standard user-interface implementations that can be placed onto the skeleton. Our job as programmer is to fill in the rest of skeleton-those things that are specific to our application. We can get a head start by using AppWizard to create the files for a very through starter application. We use the Microsoft visual C++ resource editors to design our user-interface elements visually, classwizard to connect those elements to code, and the class library to implement our application specific logic.

Version 3.0 and later of the MFC framework supports 32-bit programming for win 32 platforms, including Microsoft Windows 98 and Microsoft Windows NT version 3.5 and later. MFC win 32 support includes multithreading.

6.1.4 WINMAIN FUNCTION:

Like any program for Windows, our framework application has a winmain function. In a framework application, however, we don't write winmain. It is supplied by class library and is called when the application starts up. winmain perform standard services such as registering window classes. Then it calls member functions of the application object to initialize and run the application.

To initialize the application, winmain calls our application object's Init application and Initinstance member functions. To run the application's message loop, winmain calls the Run member function. On termination, winmain calls the application object's ExitInstance member function. Figure below shows the sequence of execution in a framework application.

Sequence of execution

Winmain standard function supplied by framework

Calls

→InitInstance initializes current instance of the application

Calls

→ RUN runs the message loop and onIdle

Calls

→ ExitInstance cleans up after the application

Note names(Winmain, Run, onIdle) the Microsoft foundation class library and Visual C++ supply these elements.

CwinApp and Appwizard:

When it creates a skeleton application, AppWizard declares an application class derived from CwinApp. AppWizard also generates an implementation file that contains the following items:

A message map for the application class

An empty class constructor

A variable that declares the one and only object of the class

A standard implementation of our InitInstance member function

The application class is placed in the project header and main source files. The name of the class and files created are based on the project name we supply in AppWizard. The easiest way to view the code for these classes is through the class view in the project workspace window.

The standard implementation and message map supplied are adequate for many purposes, but we can modify them as needed. The most interesting of these implementations is the InitInstance member function. Typically we will add code to the skeleton implementation of InitInstance.

Overridable C WinApp member functions

CwinApp provides several key overridable member functions:

InitInstance

Run

ExitInstance

OnIdle

The only CwinApp member function that we must override is InitInstance.

InitInstance member function:

Windows allows us to run more than one copy, or "Instance", of the same application. Winmain calls InitInstance every time a new instance of the application starts.

The standard InitInstance Implementation created by AppWizard perform the following tasks:

As its central action, creates the document templates that, in turn, create documents, views, and frame windows, for a description of this process, see document templates.

Load standard file options from an .INI file or the windows registry, including the names of the most recently used files.

Registers one or more document templates.

For an MDI application, creates a main frame window.

Processes the command line to open a document specified on the command line or to open a new empty document.

We can add our own initialization code or modify the code written by the wizard.

6.1.5 Run Member Function:

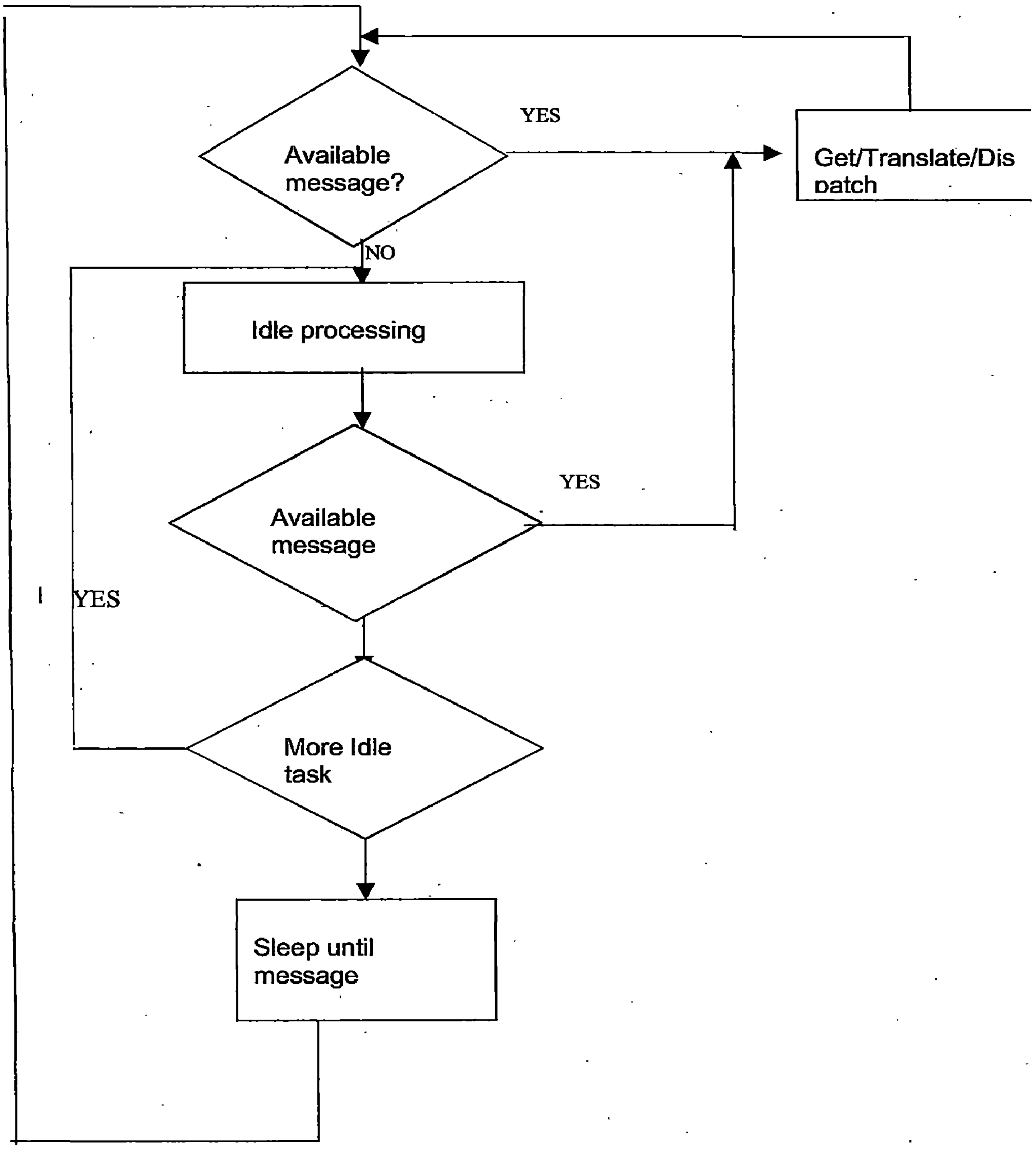
A framework application spends most of its time in the Run member function of the class CwinApp. After initialization, winmain calls run to process ~~the~~ message loop run cycles through a message loop, checking the message queue for available messages. If a message is available, run dispatches it for action. If no message is available – often the case – run calls onIdle to do any Idle – time processing that the frame or we work may need done. If there are no messages and no Idle processing to do, the application waits until something happens. When the application terminates, run calls exit Instance. Figure below shows the sequence of actions in the message loop. Message dispatching depends on the kind of message.

Exitinstance Member Function:

The exit Instance member function of class CwinApp is called each time a copy of our application terminates usually as a result of the user quitting the application. Override exit instance if we need special clean up processing, such as freeing graphics device interface (U.D.I.) resources or deallocating memory used during program execution clean up of standard items such as documents and views, however, is provided by the frame work. With other overridable function for doing special clean up specific to those objects.

On Idle Member Function:

When no Windows message are being processed. The framework calls the CwinApp member function On Idle overrides On Idle to perform background task. The default version updates the state of user interface objects such as toolbar buttons and performs clean up of temporary objects created by the frame work in the course of its operation figure below illustrates how the message loop calls On Idle when there are no.


Available message?
YES
Get/Translate/Dis patch
NO
Idle processing
Available message
YES
YES
More Idle task
Sleep until message

Besides running the Message loop and giving us an opportunity to initialize the application and clean up after it, CwinApp proves several other services.

Shell Registration

By default, App wizard makes it possible for the user to open data files that our application has created by double clicking them in the windows file manager. If our application is an MDI application and we specify on extension for the files our application creates, APP wizard adds calls to the Enableshellopen and RegistershellfileTypes member functions of CwinApp to the InitInstance override that it writes for us.

Registershellfile Types register our applications document types with file manager. The function adds entries to the registration database that windows maintain. The entries register each document type, associate a file extention with the file type, specify a command line to open the application, and specify a dynamic data exchange (DDE) command to open a document of that type.

Enableshellopen completes the process by allowing our application to receive DDE commands from file manager to open the file chosen by the user.

This automatic registration support in CwinApp eliminates the need to ship a .REG file with our application or to do special installation work.

6.1.6 File Manager Drag and Drop:

Windows version 3.1 and later allows the user to drag filenames from the file view window in the file manager and drop them into a window in our application. We might, for example, allow the user to drag one or more file names into an MDI application's main window, where the application could receive the file names and open MDI child windows for those files.

To enable file drag or drop in our application, AppWizard writes a call to the Cwnd member function that Drag Accept files for our mainframe window in our InitInstance. We can remove that call if we do not want to implement the drag-and-drop feature.

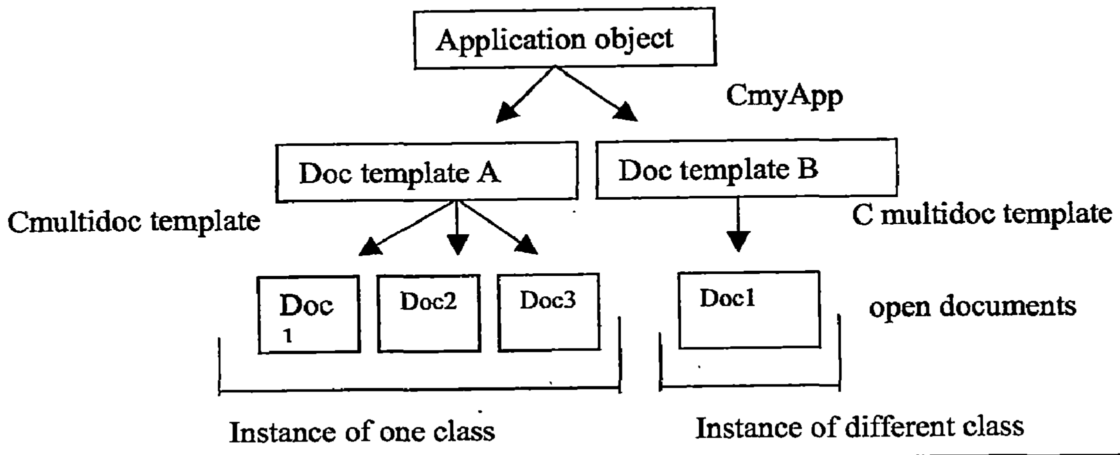
Note we can also implement more general drag-and-drop capabilities – dragging data between or within documents using OLE.

6.1.7 Keeping Track of the most recently used documents:

As the user opens or close files, The application object keeps track of the four most recently used files. The names of these files are adding to the file menu and update when they change. The frame work store these file names in an .INI file with the same name as our project and reads them from the file when our application starts up. The InitInstance override that AppWizard creates for us includes a call to the CwinApp member function Load std profile setting, which loads information from the .INI file, including the most recently used file names.
To manage the complex process of creating documents with their associated views and frame windows, the frame work uses two document template classes: C single Doc template for SDI application and C multi Doc template for MDI applications.

A CsingleDoctemplate can create and store one document of one type at a time. A CmultiDoctemplate keeps a list of many open documents of one type.
Some application supports multiple document types. For example, an application might support text documents and graphics documents. In such an application, when the user chooses the new command on the file menu, a dialog box shows a list of possible new document types to open. For each supported document type, the application uses a distinct document template object. Figure below illustrates the configuration of an MDI application that supports two document types. The figure shows several open documents.

An MDI application with two documents types



Document templates are created and maintained by the application object. One of the key tasks performed during our application's InitInstance function is to construct one or more document templates of the appropriate kind. This feature is described in Document template creation. The application object stores a pointer to each document template in its template list and provides an interface for adding document templates.

If we need to support two or more document types, we must add an extra call to Add Doc Template for each document type.

An icon is registered for each document template based on its position's in the application's list of document templates. The order of the document templates is determined by the order they are added with calls to Add Doc Template. MFC assumes that the first Icon resource in the application is the application icon, the next icon resource is the first document icon, so on.

For example, suppose a document template is the third of three for the application: If there is an icon resource in the application at index 3, that icon is used for the document template. If not, the icon at index 0 is used as default.

6.1.8 DRAWING TOOLS:

Windows provides a variety of drawing tools to use in device contexts. It provides pens to draw lines, brushes to fill interior, and fonts to draw text. MFC provides graphics –object classes equivalent to the drawing tools in windows. Table below shows the available classes and the equivalent windows GDI handle types.

The general literature on programming for the windows GDI applies to the Microsoft foundation classes that encapsulate GDI graphic objects. This section explains the use of these graphic-object classes:

Class	**windows handle type**
Cpen	**PEN**

Cbrush	**HBRUSH**
Cfont	**FONT**
Cbitmap	**HBITMAP**
Cpalette	**HPALETTE**
Crgn	**RGN**

each of the graphic object classes in the class library has a constructor that allows we to create graphic objects of that class, which we must then initialize with the appropriate create function, such as create pen.

The following four steps are typically used when we need a graphic object for a drawing operation:

Define a graphic object on the stack frame. Initializes the object with the type specific creates function, such as create pen. alternatively, initialize the object in the constructor. see the discussion of one stage and two stage creation below

Select the object into the current device context, saving the old graphic object that was selected before.

When done with the current graphics object, select the old graphic object back into the device context to restore its state.

Allow the frame-allocated graphic object to be deleted automatically when the scope is exited.

Note if we will be using a graphic object repeatedly, we can allocate it once and select it into a device context each time it is needed. be sure to delete such an object when we no longer need it.

We have a choice between to techniques for creating graphic objects:

One-stage construction: Construct and initialize the object in one stage, all with the constructor.

Two-stage construction: Construct and initialize the object into two separate stages. the constructor creates the object and an initialization function initializes it.

Two-stage construction is always safer instead of one stage-construction, the constructor could throw an exception if we provide incorrect arguments or memory allocation fails. that problem is avoided by two-stage construction, although we do have to check for failure. In either case , destroying the object is the same process.

6.1.9 MESSAGE HANDLER FUNCTION

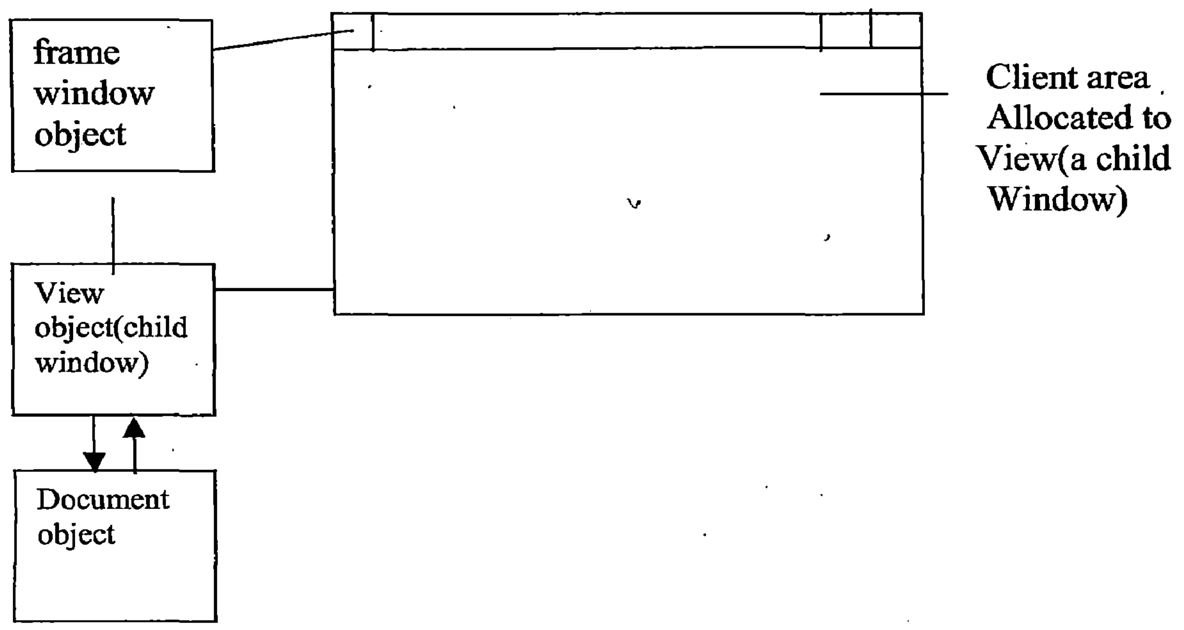
In MFC, a dedicated handler function processes for each separate message. Message-handler functions are member function of a class. some kinds of message handlers are also called " command handler ."

Writing a message handlers accounts for a large proportion of our work in writing a framework application.

What does the handler for a message do ?. The answer is that it does whatever we want done in response to that message. Class wizard will and the contents that it frames. A document frame window can be a Single document interface (SDI) frame window or a multiple document interface (MDI) child window. windows manage most of the user's interaction with the frame window: Moving and resizing the window, closing it, and minimizing and maximizing it. we manage the contents inside the frame.

The framework uses frame windows to contain views. the two components (frame and contents) are represented and managed by two different classes in MFC. A frame-window class manages the frame, and a view class manages the contents. the view window is the child of the frame window. drawing and other user interaction with the document take place in the view's client area, not the frame window's client area. the frame window provides a visible frame around a view, complete with a caption bar and standard window controls such as control menu, buttons to minimize and maximize the window, and controls for resizing the window. the "contents" consist of the windows client area, which is fully occupied by a child window – the view figure below shows the relationship between a frame window and view.

Figure: Frame window and view


frame window object
View object(child window)
Document object
Client area Allocated to View(a child Window)

The part of the framework most visible both to the user and to us, the programmer is the document and view. most of our work in developing an application with the framework goes into writing our document and view classes. this section describes: The purpose of documents and views are how they interact in the framework what we must do to implement them.

The Cdocument class provides the basic functionality for programmer-defined document classes. A document represents the unit of data that the user typically opens with the open command on the file menu and saves with the save command on the file menu.

The Cview class provides the basic functionality for programmer-defined view classes. A view is attached to a document and acts as an intermediary between the document and the user: the view renders an image of the document on the screen and interprets user input as operation upon the document. the view also renders the image for both printing and print preview.

The document/view implementation in the class library separates the data itself from its display and from user operation's 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.

A document template, as described in Document/view creation, creates documents, their associated views, and the frame windows that frame the views. the document template is responsible for creating and managing all documents of one document type.

Microsoft windows implement device-independent display. this means that the same drawing calls, made through a device context passed to our views on draw member function, and is used to draw on the screen and on other device, such as printers. we use the device context to call graphics device interface (GDI) functions, and the device driver associated with the particular device translates the calls into calls that the device can understand.

When our framework documents prints, OnDraw receive a different kind of device-context object as its argument; instead of a CpaintDc object, it gets a CDC object associated with the current printer . OnDraw makes exactly the same call through the device context as it does for rendering our document on the screen.

WIN 32

SOFTWARE DEVELOPMENT KIT

6.2 PROGRAMING WINDOWS: WIN 32 SDK...

A window in an application written for the Microsoft window operating system is a rectangular area of the screen where the application displays output and receives input from the user. A window shares the screen with other windows, including those from other applications. only one window at a time can receive input from the user. the user can use the mouse, keyboard, or other input device to interact with this window and the application that owns it.

Windows are the primary means a graphical win 32-based application has to interact with user and accomplish tasks, so one of the first tasks of a graphical win 32-based application is to create a window. this overview describes the elements of the Microsoft win 32 application programming interface (API) that application use to create and use windows; manage relationship between windows: and size, move, and display windows.

The desktop window uses a bitmap to paint the background of the screen. the pattern created by the bitmap is called the desktop wallpaper. by default, the desktop window uses the bitmap from a .BMP file specified in the registry as the desktop wallpaper the **Get Desktop Window** function returns the handle of the desktop window.

A system configuration application, such as a control panel applet, changes the desktop wallpaper by using the **System Parameters Info** function with the waction parameter set to SPI_SET DESKWALLPAPER and the lpv param parameter specifying a bitmap filename. **system parameters Info** then loads the bitmap from the specified file, uses the bitmap to paint the background of the screen, and enters the new file name in the registry.

Every graphical win 32-based application creates at least one window, called main window, that serves as the main window for the application. this window serves as the primary interface between the user and the application. most applications also create other windows, either directly or indirectly to perform tasks related to main window. each window plays a part in displaying output and receiving input from the user.

When we start an application, the system also associates a taskbar button with the application. the taskbar button contains the program icon and title. when the application is active, its taskbar button is displayed in the pushed state.

An application window includes elements such as title bar, a menu bar, the window menu (Formerly known as the system menu), the minimize button, the maximize button, the restore button, the close button, a sizing border, a client area, a horizontal scroll bar, and vertical scroll bar, an application main window typically includes all these components. the following illustration shows these components in a typical main window.

The title bar displays an application-defined icon and line of text, typically, the text specifies the name of the application or indicates the purpose of the window. An application specifies the icon and text when creating the window. the title bar also makes it possible for the user to move the window by using a mouse or other pointing device.

Most applications include a menu bar that lists the commands supported by the application. Items in the menu bar represent the main categories of commands. choosing an Item from the menu bar typically opens a pop-up menu whose items correspond to the tasks within a given category. by selecting a command, the user directs the application to carry out a task.

The window menu is created and managed by windows. It contains a standard set of menu items that when chosen by the user, set a window's size or portion, close the application, or perform tasks.

When we click the maximize or minimize button, this affects the size and position of the window. when the user click the maximize button, windows enlarges the window to the size of the screen and position the window, so it covers the entire desktop, menus the taskbar. at the same time, windows replaces the maximize button with the restore button. the restore button is a bitmap that, when clicked, restore the window to its previous size and position.

When the user clicks the minimize button, windows reduce the window to the size of its taskbar button, position the window over the taskbar button, and display the taskbar button in its normal state. to restore the application on to its previous size and position, click its taskbar button.

The sizing border is an area around the perimeter of the window that enables the user to size the window by using a mouse or other pointing device.

The client area is the part of window where the application display output, such as text or graphics. for example, a desktop publishing application displays the current page of a document in the client area. the application must provide a function called a window procedure, to process input to the window and display output in the client area.

The horizontal scrollbar and vertical scrollbar convert mouse or keyboard input into values that an application uses to shift the contents of the client area either horizontally or vertically. For example, a word-processing application that displays a lengthy document typically provides a vertical scrollbar to enable the user to page up and down through the document.

The title bar, menu bar, window menu, minimize and maximize buttons, sizing border, and scroll bars are referred to collectively as the window's non client area. Windows manages most aspects of non-client area; the application manages everything else about the window. In particular, the application manages the appearance and behavior of the client area.

An application uses several types of windows in addition to its main window, including controls, dialog boxes, and message boxes.

A control is a window that an application uses to obtain a specific piece of information from the user, such as the name of a file to open or the desired point size of a text selection. Applications also use controls to obtain information needed to control a particular feature of an application. For example, a Word-processing application typically provides a control to let the user turn word wrapping on and off.

Controls are always used in conjunction with another window ¾ typically, a dialog box. A dialog box is a window that contains one or more controls. An application uses a dialog box to prompt the user for input needed to complete a command. For example, an application that includes a command to open a file would display a dialog box that includes controls in which the user specifies a path and file name.

A message box is a window that displays a note, caution, or warning to the user. For example, a message box can inform the user to a problem the application has encountered while performing a task.

Dialog boxes and message boxes do not typically use the same set of window components, as does a main window. Most have a title bar, a window menu, a border (non-sizing), and a client area, but they typically do not have a menu bar, minimize and maximize buttons, or scroll bars.

An application must provide the following information when creating a window:

Window class

Window name

Window style

Parent or owner window

Size

Location

Position

Child-window identifier or menu handle

Instance handle

Creating data

These attributes are described in the following section.

6.2.1 Window class:

Every window belongs to window class. An application must register a window class before creating any windows of that class. The window class defines most aspects of a window's appearance and behavior. The chief component of a window class is the window procedure, a function that receive and processes all input or requests sent to the window. Windows provides the input and requests in the form of message.

6.2.2 Window name:

A window can have a name. A window name (also called window text) is a text string that identifies a window for the user. A main window, dialog box, or message box typically displays its window name in its title bar, if present. For control, the appearance of the window name depends on the control's class. A button, edit control, or static control displays its window name within the rectangle occupied by the control. A list box, combo box, or static control does not display its window name.

An application uses the **Set Window Text** function to change the window name after creating the window. It uses the **Get Window Text Length** and **Get Window Text** functions to retrieve the current window-name text from a window.

6.2.3 Window style:

Every window has one or more window styles. A window style is named constant that defines an aspect of the window's appearance and behavior that is not specified by the window's class. For example, the SCROLLBAR class creates a scroll bar control, but the SBS_HORZ and SBS_VERT styles determine whether a horizontal or vertical scroll bar control is created. A few window styles apply to all windows, but most apply to windows of specific window classes, windows and, to same extent, the window procedure for the class, interpret the style.

6.2.4 Parent or Owner Window:

A window can have a parent window. A window that has a parent is called a child window. The parent window provides the coordinate system used for positioning a child window. Having a parent window affects aspects of a window's appearance: for example, a child window is clipped so that no part of the child window can appear outside the borders of its parent window. A window that has no parent, or whose parent is the desktop window, is called a top-level window. / An application uses the **Enum Windows** function to obtain the handle of each of its top-level windows. **Enum windows** passes the handle of each top-level window, in turn, to an application-defined callback function, **Enum Windows Proc.**

A window can own, or be owned by, another window. An owned window always appears in front of its owner window, is hidden when its owner window is minimized, and is destroyed when its owner window is destroyed.

6.2.5 Location, Size and Position in the Z order:

Every window has a location, size and position in the Z order. The location is the coordinates of the window's upper left corner, relative to the upper left corner of the screen or, in the case of a child window, the upper left corner of the parent's client area. A window's size is its width and height measured in pixels. A window's position in the Z order is the position of the window in a stack of overlapping windows. For more information, see Z order.

6.2.6 Child-Window Identifier or Menu Handle:

A child window can have a child-window identifier, a unique, application-defined value associated with the child window. Child-window identifier is especially useful in applications that create multiple child windows. When creating a child window, an application specifies the identifier of the child window. After creating the window, the application can change the window's identifier by using the **Set Window Long** function, or it can retrieve the identifier by using the **Get Window Long** function.

Every window, except a child window, can have a menu. An application can include a menu by providing a menu handle either when registering the window's class or when creating the window.

6.2.7 Instance Handle:

Every win 32-based application has an Instance handle associated with it. Windows provides the instance handle to an application when the application starts. Because it can run multiple copies of the same application, windows use instance handles internally to distinguish one instance of an application from another. The application must specify the instance handle in many different windows, including those that create windows.

6.2.8 Creation Data:

Every window can have application-defined creation data associated with it. When the window is a first created, window passes a pointer to the data on to the window procedure of the window being created. The window procedure uses the data to initialize application-defined variables.

A caret is a flashing line, block, or bitmap in the client area of a window. The caret typically indicates the place at which text or graphics will be inserted

Because only one window at a time can have the keyboard focus or be active, there is only one caret in the system. Generally, each window that accepts keyboard input must create the caret when it receive the keyboard focus and destroy the caret when it loses the keyboard focus.

An application written for Microsoft windows can create a caret, display or hide it. Relocate the caret, and change its blink time.

The **clipboard** is a set of functions and messages that enable applications designed for the Microsoft win 32 application programming interface (API) to transfer data. Because all application has access to the clipboard, data can be easily transferred between applications or within an application.

A **Cursor** is a small picture whose location on the screen is controlled by a pointing device, such as mouse, pen, or trackball. (In the remainder of this section, the term mouse refers to any pointing device.) When the user moves the mouse Microsoft windows moves the cursor accordingly. Microsoft Win 32 cursor functions enable applications to create, load, display, move, confine, and destroy cursors.

In Microsoft windows, a **dialog box** is a temporary window an application creates to retrieve user input. An application typically uses dialog boxes to prompt the user for additional information for commands. A dialog box usually contains one or more controls (child window) with which the user enters text, chooses options, or directs the action of the command.

A **hook** is a point in the Microsoft windows Message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of message before they reach the target window procedure.

This overview describes window hooks and explains how to use them in a windows-based application.

An **Icon** is a picture that consists of bitmapped image combined with a mask to create transparent areas in the picture. This overview describes creating, displaying, destroying, and duplicating icons.

In Microsoft windows, a **Keyboard accelerator** (or simply accelerator) is a keystroke or combination of keystrokes that generates a **WM-COMMAND** or **WM-SYSCOMMAND** message for an application.

A **Menu** is a list of menu items. Choosing a menu item opens a submenu or causes the application to carry out a command.

The Multiple document interface (MDI) is a specification that defines a user interface for applications that enable the user to work with more than one document at the same time.

This overview describes the structure of an MDI application and how to take advantage of the built-in MDI support found in the Microsoft win 32-application program interface (API).

MDI is an application-oriented model. Many new and intermediate users find it difficult to learn to use MDI applications. In the future, application will use a more document-oriented model. Therefore, we may want to consider a SDI model for our user interface. However, until there is an alternative that fully replaces the MDI model, we can use MDI for applications, which do not easily fit into another model.

A **Resource** is binary data that a resource compiles or developer adds to an application's executable file. A resource can be either standard or defined. The data in standard resource describes an icon, cursor, menu, dialog box, bitmap, enhanced meta file, font, accelerator table, message-table entry, or version. An application-defined resource, also called a custom resource, contains any data required by a specific application.

A **Timer** is an internal routine that repeatedly measures a specified interval, in milliseconds. Each time the interval (or time-out value) elapses, the system notifies the window associated with the timer, because the accuracy of a timer depends on the system clock rate and how often the application retrieves messages from the message queue; the time-out value is only approximate.

Every window is a member of a window class. A window class is a set of attributes that Microsoft window uses as a template to create a window in an application. This overview describes the types of window classes, how windows locate them, and the elements that define the default behavior of windows that belong to them.

In Microsoft windows, every window has an associated window procedure ¾ a function that processes all messages sent or posted to all windows of the class, all aspects of a window's appearance and behavior depends on the window procedure's response to these messages.

A **window property** is any data assigned to a window by the **setprop** function. A window property is usually a handle of the window-specific data, but it may be any 32-bit value. Each window property is identified by a string name.

Common Controls are a set of windows that are supported by the common control library, which is a dynamic-link library (DLL) included with the Microsoft windows operating system. Like other control windows, a common control is a child window that an application uses in conjunction with another window to perform input and output (I/O) tasks.

An **Animation control** is a window that silently displays an audio video Interleaved (AVI) clip. An AVI clip is a series of bitmap frames like a movie. Although AVI clips can have sound, We can not use such clips with animation controls. We can use only silent AVI clips.

A **Drag list box** is a special type of list box that enables the user to drag items from one position to another. An application can use a drag list box to display strings in a particular sequence and allow the user to change the sequence.

A **Header control** is a window that is usually positioned above columns of text or numbers. It contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column.

A **Hot-key control** is a window that enables the user to enter a combination of keystrokes to be used as a hot key. A hot key is a key combination that the user can press to perform an action quickly. (For example, a user can create a hot key that activates a given window and brings it to the top of the Z order.) The hot-key control displays the user's choice and ensures that the user selects a valid key combination.

An **Image list** is a collection of same-sized images, each of which can be referred to by its index. Image lists are used to efficiently manage large sets of icons or bitmaps. All images in the image list are contained in a single, wide bitmap in screen device format. An image list may also include a monochrome bitmap that contains masks used to draw images transparently (icon style).

A **List view control** is a window that displays a collection of items, each item consisting of an icon and a label. List view controls provide several ways of arranging items and displaying individual items. For example, additional information about each item can be displayed in columns to the right of the icon and label.

A **progress bar** is a window that an application can use to indicate the progress of lengthy operation. It consist of a rectangle that is gradually filled, from left to right, with the system highlight color as an operation progresses.

A **Property Sheet** is a window that allows the user to view and edit the properties of an item. For example, a spreadsheet application can use a property sheet to allow the user to set the font and border properties of a cell or to view and set the properties of a device, such as disk drive, printer or mouse. A property sheet contains one or more overlapping child windows called pages, each containing control windows for setting a group of related properties. For example, a page can contain the controls for setting the font properties of an item, including the type style, point size, color, and so on. Each page has a tab that the user can select to bring the page to the foreground of the property sheet.

A **status window** is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status window can be divided into parts to display more then one type of information.

A **Toolbar** is a control window that contains one or more buttons. Each buttons send a command message to the parent window when the user chooses it. Typically, The buttons in a toolbar correspond to item in the application's menu, providing an additional and more direct way for the user to access applications commands.

A toolbar has built-in customization feature, including a system- defined customization dialog box that allow the user to insert, delete, or rearrange toolbar buttons. An application determines whether the customization features are available to the user and controls the extent to which the user may customize the toolbar.

A **Tooltip control** is a small pop-up window that displays a single line of descriptive text giving the purpose of tools in an application. A tool is either a window, such as a child window or control, or an application-defined rectangular area within a window's client area. A tool tip control is hidden most of the time, appearing only when the user puts the cursor on a tool and leaves it there for approximately one-half second. The tool tip control appears near the cursor and disappears when the user clicks a mouse button or moves the cursor off of the tool. A single tool tip control can support any number of tools.

A **Trackbar** is a window that contains a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the trackbar sends notification messages to indicate the change.

A **tree-view control** is a window that displays a hierarchical list of items, such as the heading in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems.

A **Button** is a control the user can turn on or off to provide input to an application. There are several types of buttons and, within each type, one or more styles to distinguish among buttons of the same type. The user turns a button on or off by selecting it using the mouse or keyboard. Selecting a button typically changes its visual appearance and state (from checked to unchecked, for example). Windows, the button, and the application cooperate in changing the button's

appearance and state. A button can send message to its parent window, and a parent window can send messages to a button. Some buttons are painted by windows, some by the application. Buttons can be used alone or in-groups and can appear with or without application-defined text (a label). They belong to the BUTTON window class.
Although an application can use buttons in overlapped, pop-up, and child windows, they are designed for use in dialog boxes, where window standardizes their behavior. If an application uses buttons outside dialog boxes, it increases the risk that the application may behave in a non-standard fashion. Applications typically either use buttons in dialog boxes or use window sub classing to create customized buttons.

Window provides five kinds of buttons: push buttons, check boxes, radio buttons, group boxes, and owner-drawn buttons. Each type has one or more styles.

A combo box is a unique type of control, defined by COMBOBOX class that combines the much of the functionality of a list box and an edit control.

The Microsoft win 32 application-programming interface (API) provides three types of combo box: simple combo boxes (CBS_SIMPLE), drop-down combo boxes (CBS_DROPDOWN), and drop-down list boxes (CBS_DROPDOWNLIST).

There are also a number of combo box styles that define specific properties. For example, two styles enable an application to create an owner-drawn combo box, making the application responsible for displaying information in the control.

A combo box consists of a list and a selection field. The list presents the options a user can select and the selection field displays the current selection. Except in drop-down list boxes, the selection field is an edit control and can be used to enter text not in the list

Microsoft Windows provides dialog boxes and controls to support communication between the application and user. An **edit control** is a rectangular control window typically used in dialog box to permit the user to enter and edit text from the keyboard.

An edit control is selected and receives the input focus when a user clicks the mouse inside it or presses the TAB key. After it is selected, the edit control display its text (if any) and a flashing caret that indicates the insertion point. The user can then enter the text, move the insertion point or select text to be moved or deleted by using the keyboard or the mouse. An edit control can send notification message to its parent window in the form of **WM-COMMAND** messages. A parent window can send messages to an edit control in a dialog box by calling the **SendDlgItemMessage** function. Each of the messages sent to edit controls is discussed in this overview.

Windows provides both single line edit controls (some time called SLEs) and multiline edit controls (sometimes-called MLEs). Edit controls belong to the **EDIT** window class.

A combo box is a control that combines much of the functionality of an edit control and a list box. In a combo box, the edit control displays the current selection and the list box presents option a user can select.

Many developers use the dialog boxes provided in the common dialog box library (COMDLG32.DLL) to perform tasks that otherwise might require customized edits controls.

A **list box** is a control window that contains a list of items from which the user can choose.

List box items can be represented by text strings, bitmaps, or both. If the list box is not large enough to display all the list box items at once, the list box can provide a scroll bar. The user maneuvers through the list box items, scrolling the list when necessary, and selects or removes the selection from items. Selecting a list box item changes its visual appearance, usually by changing the text and background colors to the colors specified by operating system metrics for selected items. When the user selects an item or removes the selection from an item, window sends a notification message to the parent window of the list box.

A dialog box procedure is responsible for initializing and monitoring its child windows, including any list boxes. The dialog box procedure communicates with the list box by sending messages to it and by processing the notification messages sent by the list box.

A window in an application written for Microsoft windows can display a data object, such as document or bitmap that is larger than the window's client area. When provided with a **scroll bar**, the user can scroll a data object in the client area to bring into view the portions of the object that extend beyond the borders of the window.

Scroll bars should be included in any window for which the content of the client area extends beyond the window's borders. A scroll bar's orientation determines the direction in which scrolling occurs when the user operates the scroll bar. A horizontal scroll bar enables the user to scroll the content of a window to the left or right. A vertical scroll bar enables the user to scroll the content up or down.

A **Static control** is a control that enables an application to provide the user with certain types of text and graphics that typically require no response.

Windows provides an application programming interface (API) that lets we take advantage of appbar services provided by the system. The services help ensure that application-defined appbars operate smoothly with one another and with taskbar. The system maintains information about each appbar and sends the appbars message to notify them about events that can affect their size, position, and appearance.

TESTING

CHAPTER: 7

TESTING

Testing of any system is the last phase of the software development before it is delivered to the user client. Testing phase includes the following criteria, which must be satisfied for final submission of the project:

- Each phase works properly in accordance with the design phase specification, which were supposed to be achieved.
- Each phase starts at the right point of time and at the correct location.
- On each phase of the life cycle of the system software the validation criterion is working properly.
- The various tests must include : Unit testing , Integration testing, validation testing and finally system testing.

In the testing phase of "THEATRE VISION", we tested all the above phases of the system in presence of the user number of times till user was satisfied with the overall performance of the system as its own requirement.

We showed the user the following steps in the testing phase :

- Each phase is working in accordance of the user requirements
- Each screen is working properly in the system
- Options provided for are sufficient
- Online-help provided to each topic in the system is sufficient
- Format for each screen is Ok or any modification if needed is possible in future.

CONCLUSIONS:

It is concluded that this software provides a Graphical User Interface for pre-designing the light and sound show. It is a very useful software for stage shows and one have no burden about the design of show because once a show have designed we can use it very easily and one can change according to his need and get trial before the real show.

This software provides the following information
* We can test at each scene levels
* Cue designing and testing at design level
* A maximum of 1,00,000 cues giving a show design possibility of shows for up to 88 hours and beyond
* Selected from a maximum of 1000 preset design scene. From these any 25 scenes can be fade-in, fade-out or locked simultaneously
* 8 effects can be simultaneously used in a cue with the preset scene combinations (relay effects can be cloud generator, smoke generator, video projector, slide projector, audio effects, lightning generator, flashers, color generator etc.)
* Maximum 512 dimmer control
* Designed software is fully compatible for DMX-512

Based on the present work it can be concluded that this software can be efficiently and effectively used whenever a similar requirement arises.
This software has successfully tested on 26 May 2002 in "Theatre Technocraft" company in Delhi and it is tested on single light in "CNS COM SOFT SYSTEMS Pvt. Ltd." For Intelligent Dimmers.

LIMITATION

AND

FUTURE PERSPECTIVES

OF

THEATRE VISION (LIGHT CONTROL SYSTEM)

Limitations and future Perspectives of THEATRE VISION

Electronic media industry with the rapid advancement in electronics and especially with involvement of computers is producing innovative products with faster data access speeds, more accuracy, better quality output, more resistibility, etc. will pose upgrade requirements in future.

Especially with new advancements in the sound industry, like introduction of the optical discs which have a faster audio data transfer will be threats for the current version of THEATRE VISION (Light and Sound control system) since the DMX-512 protocol proves short to synchronize with it. For this implementation the protocol standard should be upgraded to DMX-*, which is an open protocol used for conveying control information. It supports data packet exchange and hence can contain more than 512 data channels using a multiplexing scheme.

Beside this, human errors in coding will be welcomed if any inherent bug not detected after all of testing and implementation come up. Though none of light and sound shows have gone beyond 100dimmers, 250 scenes and 36000 cues till date and the THEATRE VISION software supports 512dimmers, 1000 scenes and 100000 cues; still in future if this limit goes beyond these software limits, the application will need to be upgraded. Similar is the case for the number of relay effect. for a few years there is no threat of modification as regarding these areas.

Future scope

This software can be further extended for more than 512 dimmers and can be used in future for Intelligent light. Which can handle both light and sound. And this can be launched on internet by its redesigned show. which will be used for stage show and fashion show. by this every body did not need to design the show himself.

Users Manual

DMX 512 OUTPUT. 512 CHANNEL

THEATRE VISION

THEATRE TECHNOCRAFT

<u>*Introduction*</u>

Here you can make & record PC based LIGHT shows with least efforts and best accuracy. Just enjoy yourself while running the show fully automatically. Creates scenes non-sequentially helps you to sand-witch new scenes. You can learn working of this SOFTWARE by roaming through Tips.

This software is fully compliant with USITT standard for DMX 512 data transmission. It is user friendly and designed for inexperienced computer operators. One can exploit full potential of DMX 512 protocol using this software. All 512 channels can be engaged at one go. One can design/save/edit/auto run a show live and fully automatically.

One can also use it in place of a Console to operate channels manually. Up to 500 scenes can be stored in particular show. Any channel can be programmed as Chaser or Flasher. In case of power failure or during rehearsals, show can be resumed from any desired scene.

Installation

- **THEATRE VISION** System is designed to run on Pentium II+ or equivalent PC (min.) with Windows 95/98, a Sound Blaster Live! Platinum 5.1 is powered by the EMU10K1 audio processor, 'Sound Blaster' compatible sound card, a video card (MPG III), monitor capable of 16bit 600x800 color mode (or higher if your monitor allow) and a minimum of RAM (64MB). Power management functions and screensavers should be disabled on your PC as these can cause desk events to be delayed. It is also essential that your sound card and video card is working properly in Windows for *THEATRE VISION Software*. To play sound events - check that you can play sounds correctly in Windows and that your speakers are turned on

- **The Mains Power** supply given to the PC should be proper. If you are running the PC with the same mains supply as your lighting dimmer packs etc. any power surges/spikes may cause problems with the PC hardware which could result in the Theater Vision software not functioning correctly (this is of course with any software on the PC). If you find that the PC screen image is 'flickering' or sound events are not working properly then it indicates that the PC is having power problems and you will need a separate power supply or use a surge protector or UPS (Uninterrupted Power Supply) unit to run it.

- **The sound card** : THEATRE VISION recommends Sound Blaster Live! Platinum 5.1, powered by the EMU10K1 audio processor. With Platinum 5.1 SBLive Card, you will experience better effect and impact than any other sound card. ***The Microsoft DirectX system has been implemented in THEATRE VISION to enable stable multiple playback and flexible Sound & Light functions.*** To experience true-to-life quality sound, you must install sound card properly in Windows and enable the audio devices for recording and playing using Windows. You can use Audio Mixer from system tray to adjust the volume. If you already have a sound card like *THEATRE VISION*, first play and record from Windows properly, then *THEATRE VISION* will be able to play as well. If you experience any problem with audio or the sound-to-light section, please contact your manufacturer to upgrade the sound driver. After loading the driver, please confirm that sound card is working properly.

- **The video card** installed with THEATRE VISUAL is MPG -III with 3-D effect and a good color contrast for best effect.

- **To install the software** just insert the CD. The software will install and update registry entries itself. The other method is to open the CD in window's explorer and click on the file "setup.exe"

- **THEATRE VISION** System can also be installed on a Laptop PC .If you are using *THEATRE VISION* on a laptop PC with an external DMX output device, a dongle, **Dongle II (Artistic Licence (UK) Ltd.)** hardware is required with this software in order to make and run the show. The Dongle unit is to be attached to parallel port (LPT1) of the computer before using it. Then make sure that the screen mode selected on the laptop is 800X600 16-bit color. The higher 24- bit and 32- bit color depths, incorrectly installed video or mouse drivers can cause the mouse to move jerkily. Video adapters on laptops are more prone to cause mouse /speed problem than on standard desktop PCs. The normal number keys at the top of the laptop PC keyboard can be used instead of the keypad referred in the manual as few laptops possess a separate keypad. You will need to hold down the shift key however to enter [/], [*], [-] etc. whenever required.

- **To activate this software** for making and running the show, a Dongle II **(Artistic Licence (UK) Ltd),** hardware supplied with this software is required. The Dongle unit is to be attached to parallel port (LPT1) of the computer before starting it.

- **Before Software is loaded** set your computer for best view of show screen (VDU) resolution at 800X600, then click THEATRE VISION desktop icon to load software, first screen appears (Main Patch Windows) and prompt "Dongle successfully loaded". Incase Dongle is not working properly or showing some delay in response, you can set the refresh rate of Dongle data by clicking on Go menu and then click on Refresh Rate (rarely required). You can set the rate in multiples of default (ranging from 2x to 10x).

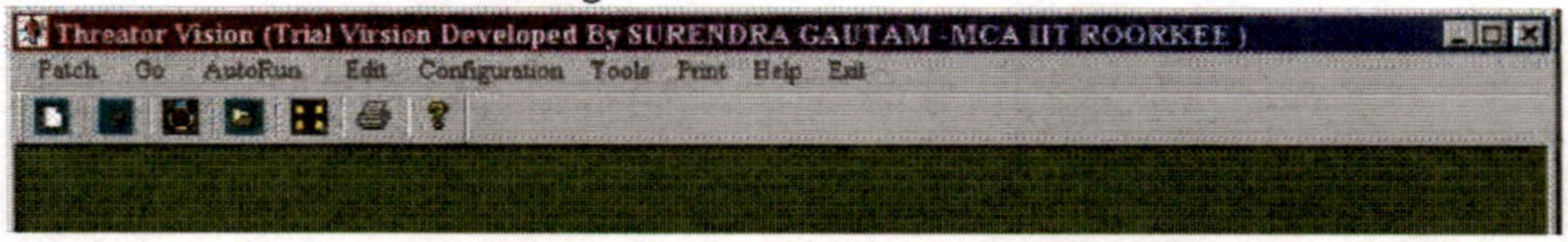
- **The main window that appears for editing the show is the heart of** THEATER VISION because the show is developed using this dialog box. In this window, we load every scene of the show one by one. Loading the scene means its scene time, set the lights that are suitable for that scene, set the Fade In and Fade Out time of the lights, set the flash lights and the flash speed and set the track that is to be tuned when the scene will be shown on the stage

THEATRE VISION Functions:

Main Window

This is the Main window having a menu bar and a toolbar.

The menu bar has the following menu


Threator Vision (Trial Virsion Developed By SURENDRA GAUTAM -MCA IIT ROORKEE)
Patch
Go
AutoRun
Edit
Configuration
Tools
Print
Help
Exit

1. **Patch**: Creates a new show for designing the scenes. Click on it and enter the Show name, which you want. The path where the show has to be stored is set default.

   When you enter name and press OK button the **design view window** will be opened and all intelligent lights and faders data will be loaded with NULL value.

Design View Window: The design view window will appear. User will design a show from this window. This window has the following design structure.

1. **Channel:** Channel from 1 to 512 are of grey colour for intelligent light. and faders.

2. **Show information:** This section has following options:

   Designer Name: Here you have to type the name of designer who is preparing the show.

   Electrician Name: The name of electrician how help in setting the intelligent lights.

   Date: Here you have to select the current date it will automatically show the date when you click the scroll list.

   Scene No.: It will automatically generate the show no. but in first time it will show Scene No. 1;

3. **Time in seconds (Intelligent Light Timing):** This section has following option.

   Fade in: Fade in time of intelligent lights

Fade out: Fade out time of intelligent lights.

Flash/Chase Speed: Flash or chase speed (0% to 100%).

4. **Previous Scene Option:** This section has two radio (add and cross) buttons and one remove button.

ADD: Add previous scene to current scene. You can not add in scene no. 1. Added scene will fade in but not fade out until CROSS scene encounter.

CROSS: No CROSS scene occurs before ADD scene. CROSS scene get fade in while all previous ADDED scenes will fade out in time (fade out) of just previous one of CROSS scene.

In case of the user click on add radio button, the existing Dimmers will not fade out, instead all the changed faders for next scene will be added as per their fade in time.

In case of user click on cross radio button, the existing Dimmers will fade in as per their fade in time and fade out all previous scene simultaneously

5. **CD Option:** It has two things to add

CD Type: There are two options

1. CD-Audio
2. Vidio-Disk

Track No.: it can accept up to 1,000 track no.

6. **Flash Pattern No.:** In this option you can choose upto Eight flash patterns .

7. **Scene Option:** This option shows Different buttons for setting scene

New: When you click **New Button** you find new scenesetting window with next scene No.

Save: After setting all the values in all options and giving appropriate values to all cannels you have to save your setting by clicking the Save button.

OK: When you save the setting and you want no more scene to create you can click **OK** button to exit the SceneSetting window.

Copy: In this option you can copy the previous scene if you want next scene with a little bit change with respect to previous scenes.

Cancel: If you do not want any change in the scene then you can click **Cancel** Button to Exit the SceneSetting window.

Paste: When you click **Copy Button** the **Paste** Button will be highlighted and you can paste the copied scene for next SceneSetting.

8. **Change Option:** In this Option you can change you SceneSetting simultaneously you do not need to reopen the SceneSetting window. You have given here four Options:-

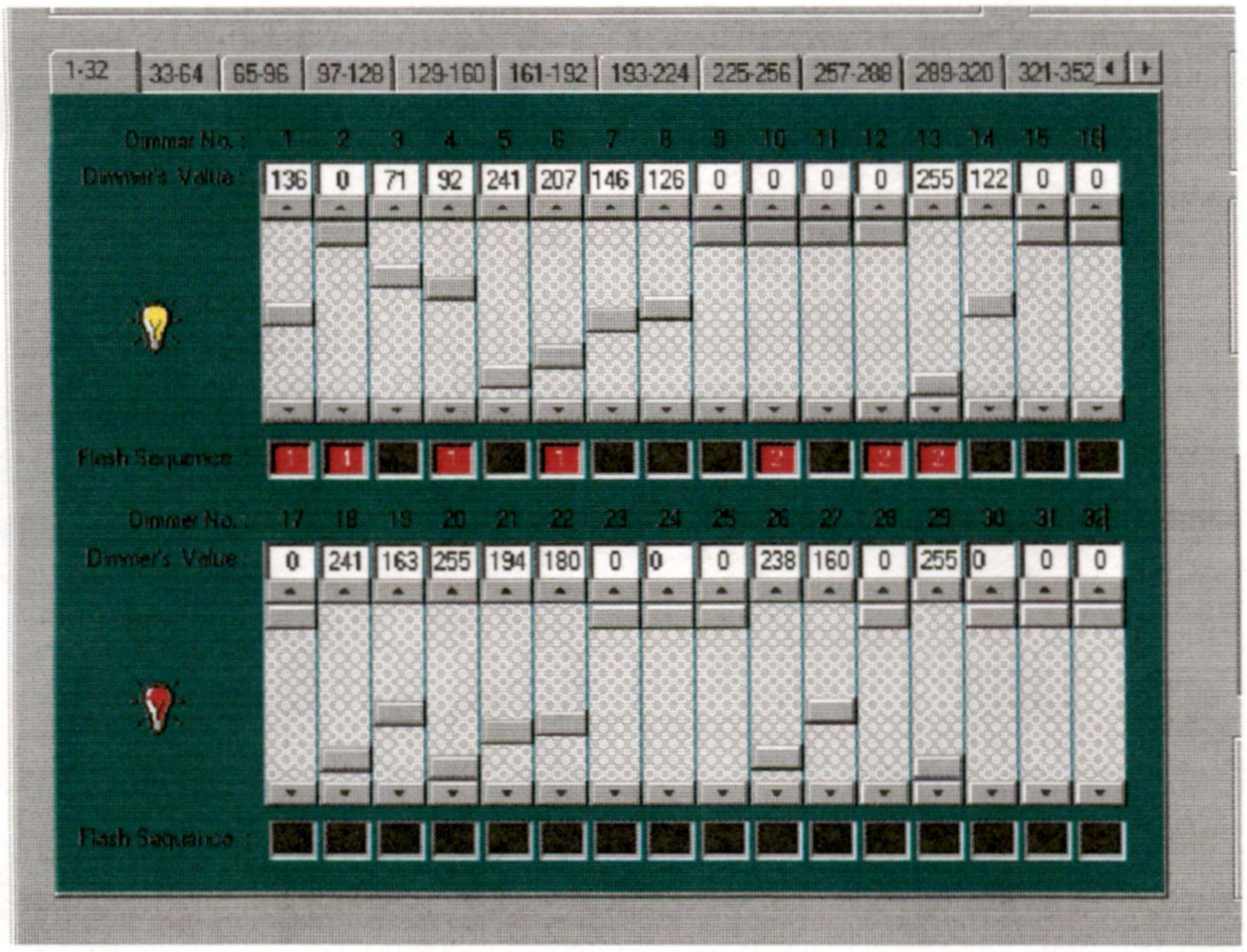
   First: Here you choose Scene No. 1

   Previous: In this button you choose the previous scene

   Last: Here you choose the last Scene created if you are in among of all other scenes.

   Next: Here you choose the Next scene

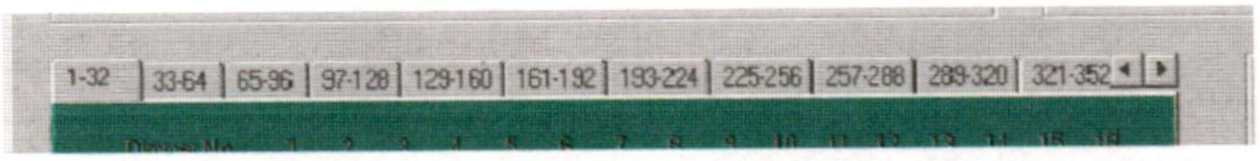
9. *Level:* These boxes show the maximum level of any channels. For Intelligent Light (1-256) can insert from Light Patcher Window.



Dimmer No. :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Dimmer's Value :	136	0	71	92	241	207	146	126	0	0	0	0	255	122	0	0

Dimmer No. :	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Dimmer's Value :	0	241	163	255	194	180	0	0	0	238	160	0	255	0	0	0

Tab Bar: In this Bar you can select different Channels just by Clicking the Tabs. We have divided it in 16 Tabs:
1 – 32,   33 – 64,   65 – 96 ,………, 449 – 480,   481 - 512

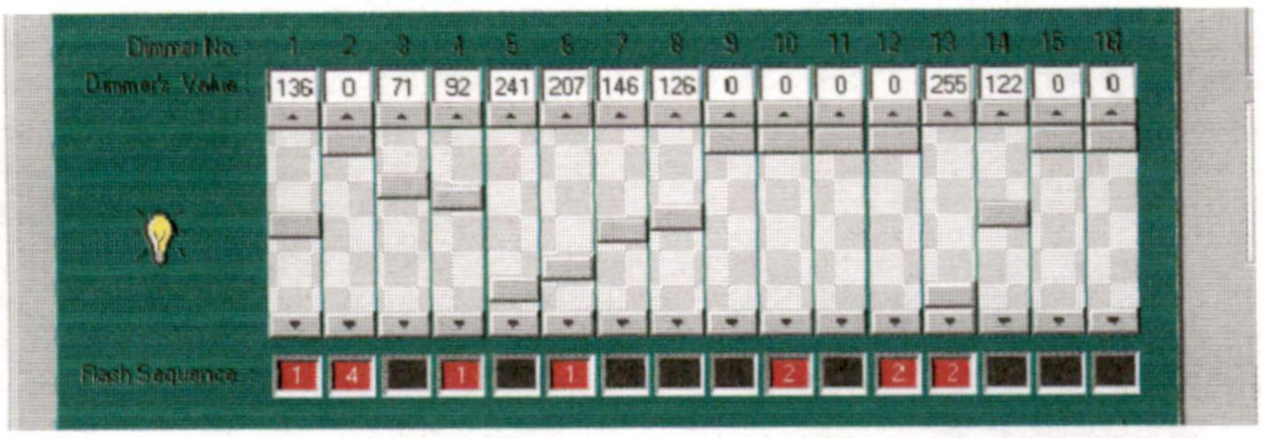


In each tab it will show save type of Screen but changed Channel No. and dimmer No.. Every Tab contains 32 dimmers and every dimmer has a Vertical Scroll Bar By Sliding it you can feed the dimmer's value in the range of 0 to 255.

Dimmer's No.: Every Tab contains 32 dimmers
1 – 32,   33 – 64,   65 – 96 ,………, 449 – 480,   481 - 512

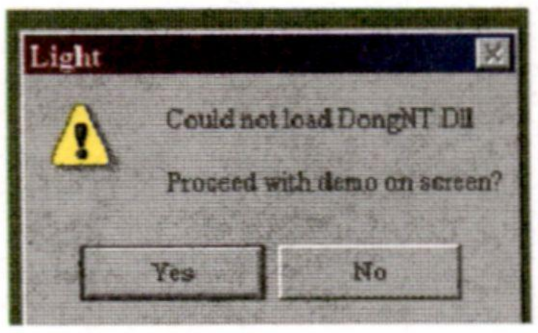
Dimmer's Value: every dimmer has a Vertical Scroll Bar By Sliding it you can feed the dimmer's value in the range of 0 to 255.

Flash Sequences: We can select Flash Sequences just by Choosing the Flash Pattern and clicking the Flash Sequence under every Dimmer which is how by Black Square and by clicking it it will show Red Square with Flash Pattern



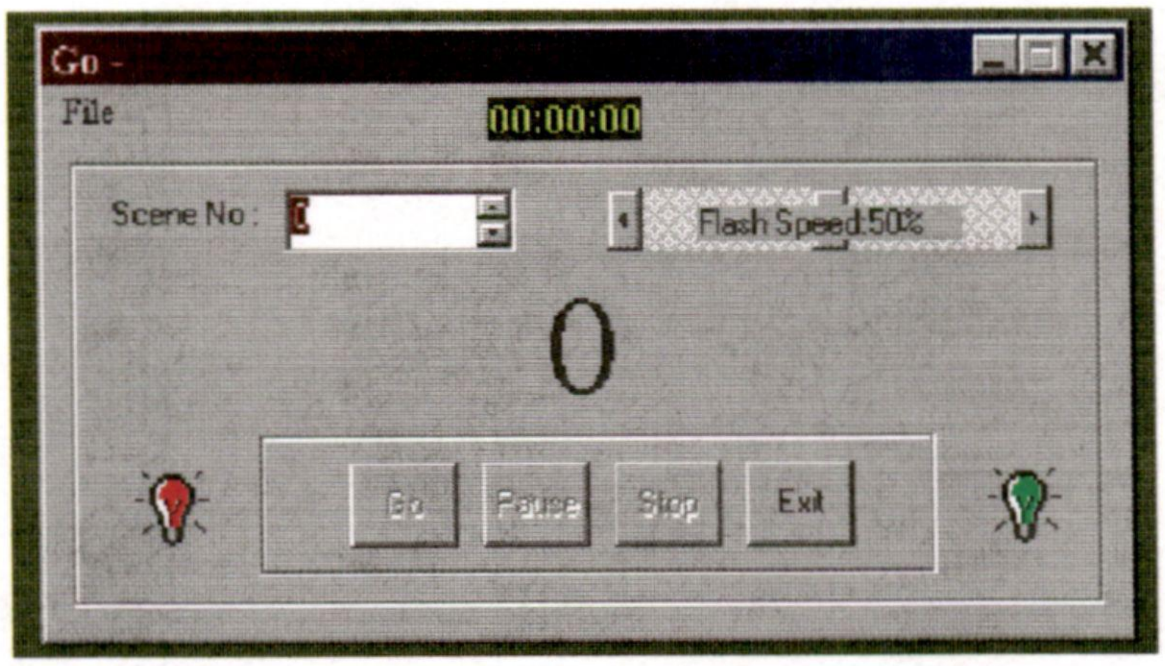
Dimmer No. :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Dimmer's Value :	136	0	71	92	241	207	146	126	0	0	0	0	255	122	0	0
Flash Sequence :	1	4	■	1	■	1	■	■	■	2	■	2	2	■	■	■

Go Menu: This has a single menu item. When you Click the Go Option you may see a dialog box if you have not installed Dongle Driver. But still you can proceed with Demo of "THEATRE VISION" show. So click "Yes" Button.

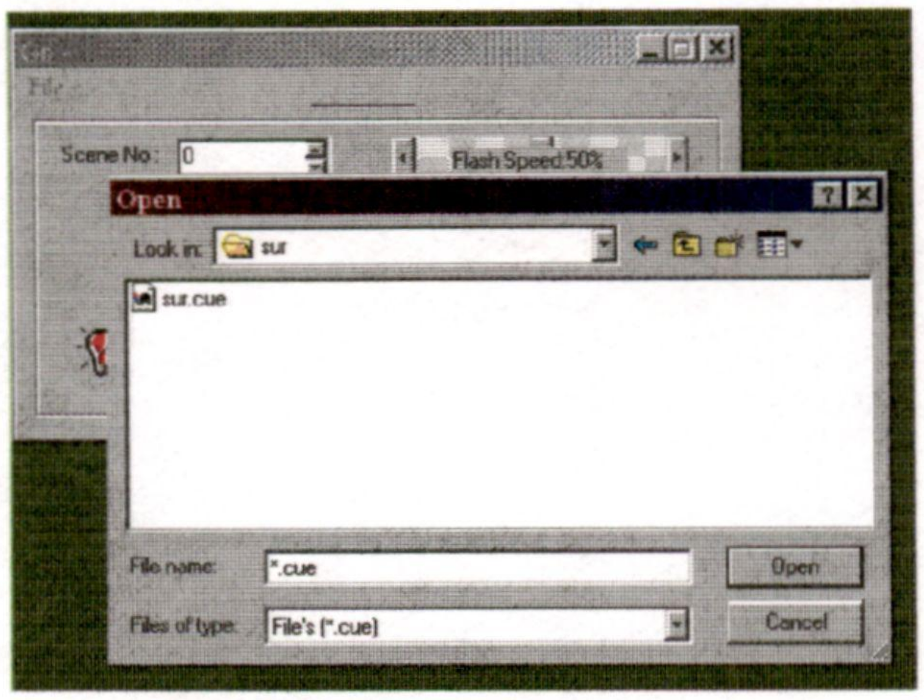

Light

Could not load DongNT Dll

Proceed with demo on screen?



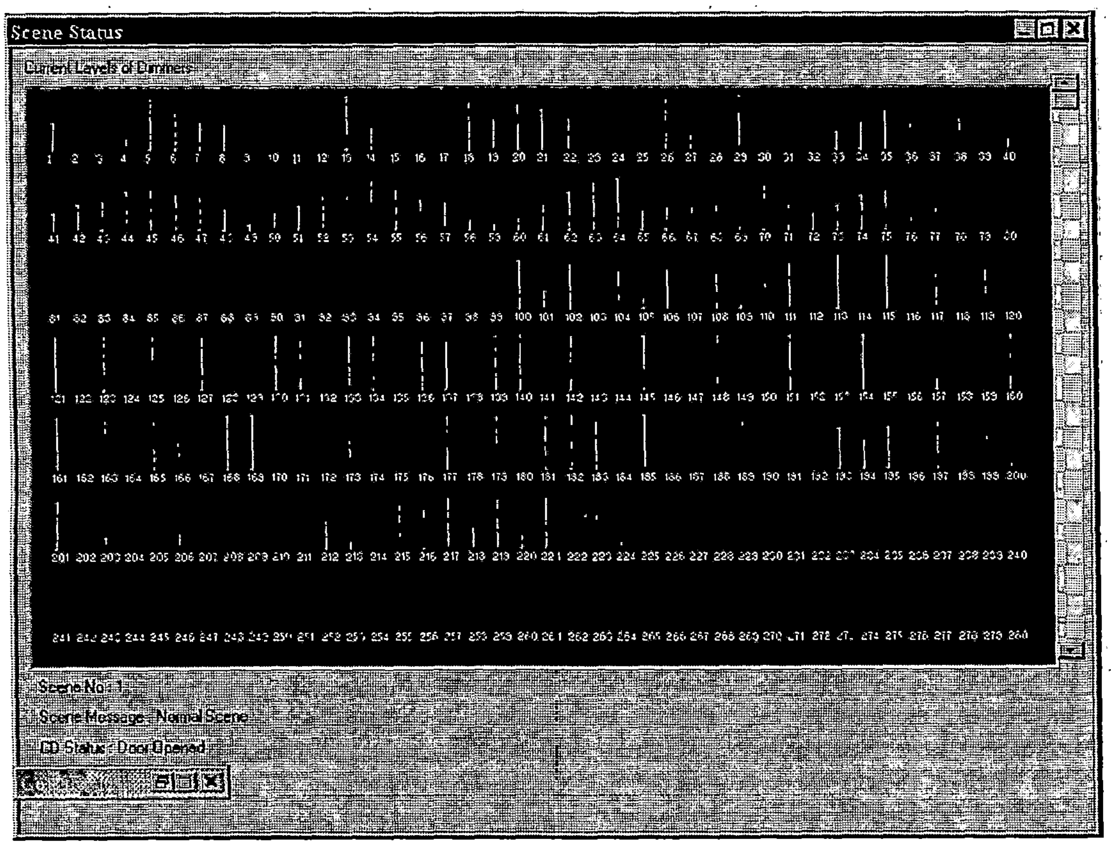
File Menu: In file Menu there are two option "Open" and "Exit". In Open menu option you open the existing show.



Scene No: you can select the scene No. which you have to Run.

Flash Speed(Horizontal Scroll Bar) : you can increase or decrease the speed of show.

Go Button: When you Click Go Button you can see the running show on this window . The background is black and dimmers light are shown in Red colours columns. The transmission of channel data appears on the status window and audio tracks or video file start playing automatically.



Scene No : 1

Scene Message : Normal Scene

CD Status : Door Opened

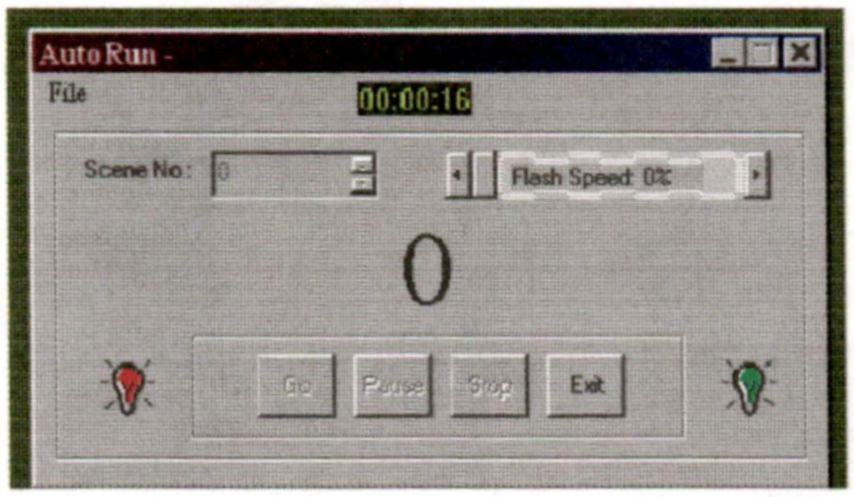
Pause button: Click on it to pause a running scene.

Stop Button: Click on it to Stop the current show.

Exit: Click on it to exit from go dialog box.

AutoRun Menu: It is a single menu item.

Click on **AutoRun** menu item to open AutoRun dialog box. It has the following options



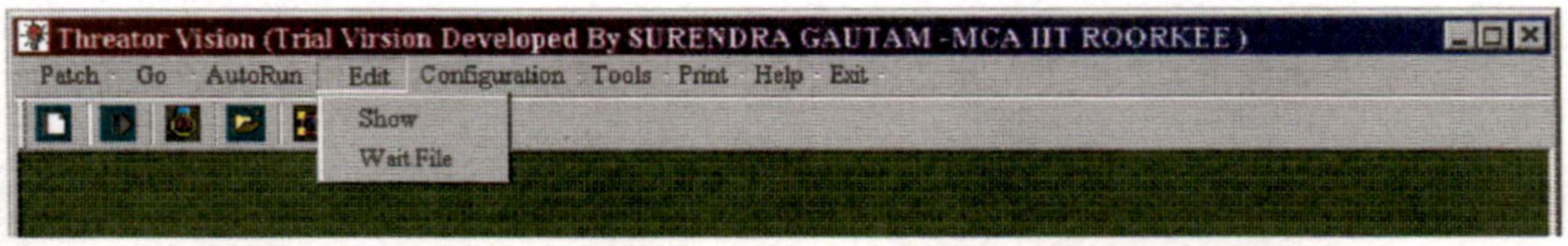
Go button: Click on it to start a show. The next scene will start automatically when previous scene finishes.

Pause: Click on it to pause a running show.

Stop: Click on stop button to stop a show.

Exit: Click on it to exit from auto run dialog box.

Edit Menu: This is a popup menu has two options


Threator Vision (Trial Virsion Developed By SURENDRA GAUTAM -MCA IIT ROORKEE)
Patch
Go
AutoRun
Edit
Configuration
Tools
Print
Help
Exit
Show
Wait File

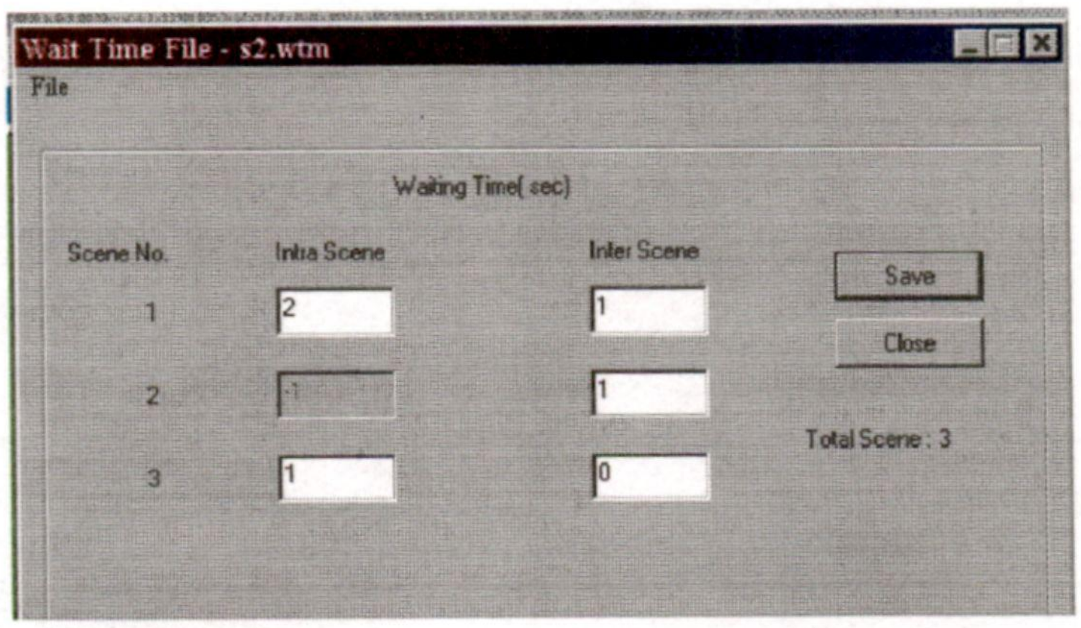
1. ***Show:*** Open an existing show. Click on it or Press Ctrl+O. The design view window will be opened and all intelligent lights and faders data will be loaded.

2. ***Wait File:*** In wait file menu option you can open .wtm file of every show. And you can edit the following

   Intra Scene

   Inter Scene

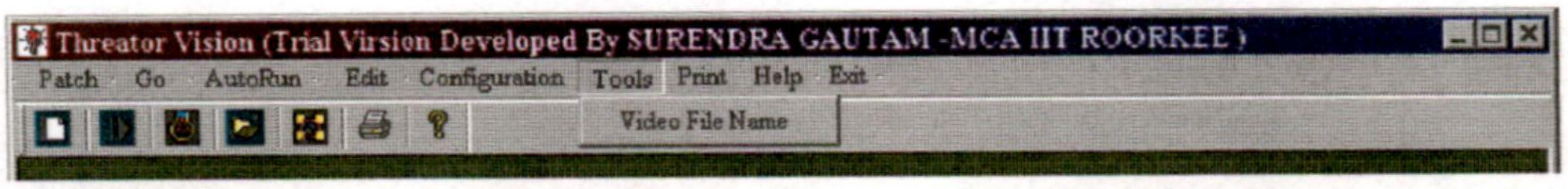
   And after editing it you can save the settings.



Scene No.	Intra Scene	Inter Scene
1	2	1
2	-1	1
3	1	0

Total Scene : 3

Tools Menu: This popup menu has single options for Vidio CD Path Name Dialog Box

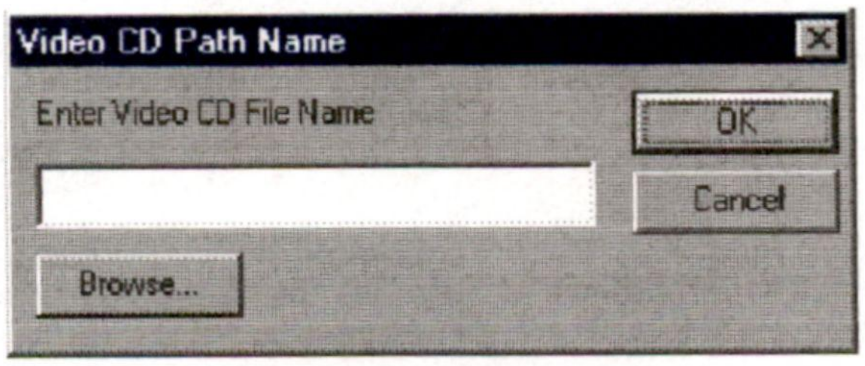

Threator Vision (Trial Virsion Developed By SURENDRA GAUTAM -MCA IIT ROORKEE)
Patch
Go
AutoRun
Edit
Configuration
Tools
Print
Help
Exit
Video File Name

Video CD: Popup menu has option::

1. Video File Name sub menu command

If you have selected in Fader dialog , CD type : Video Disc (Run Video movie with Dimmers) For that you do the following things.

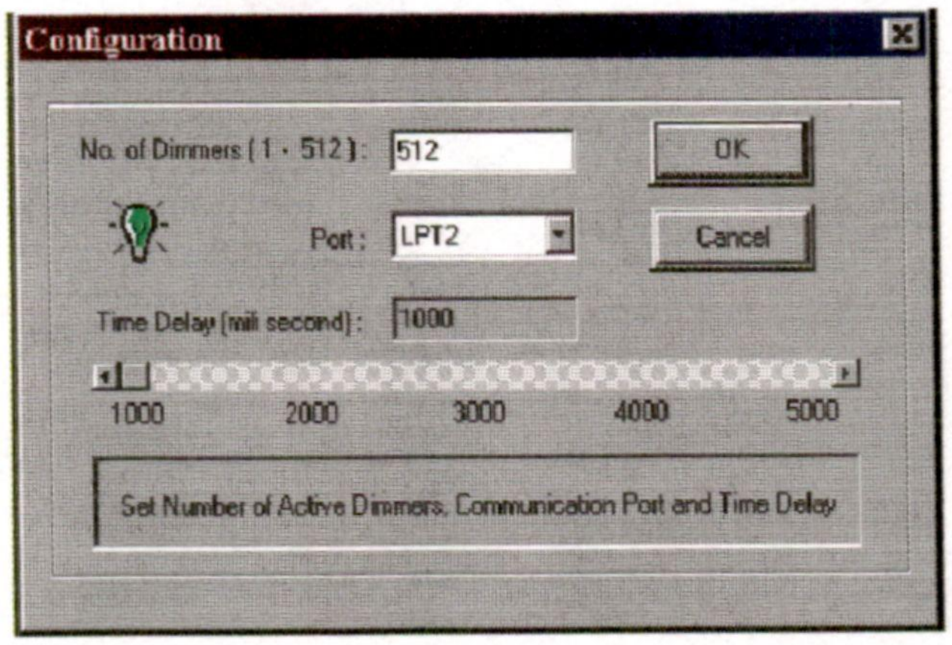
Video File Name menu command opens Video CD Path Name dialog box:



Enter the CD video file name or click on Browse button to search the CD video file name.

Click ok button.

Configuration Menu: Click on it to open configuration window.



No. of Dimmers (1 - 512): 512

Port : LPT2

Time Delay (mili second) : 1000

Set Number of Active Dimmers, Communication Port and Time Delay

The purpose of the configuration is to set the channel descriptions and parallel port.

Channel number from 1 to 512 are for intelligent light by default.

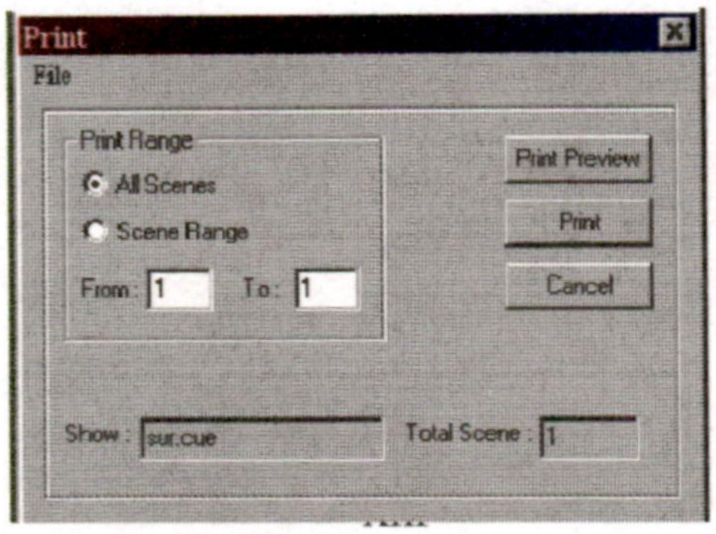
There are three options:

No. of Dimmers(1 – 512): here you have to enter the no. of dimmers which you have to set active.

Port: Choose the Parallel port for DMX-512 cable.

Time Delay (in milli seconds): Delay time from 1000 to 5000.

Print Menu:



(•) All Scenes

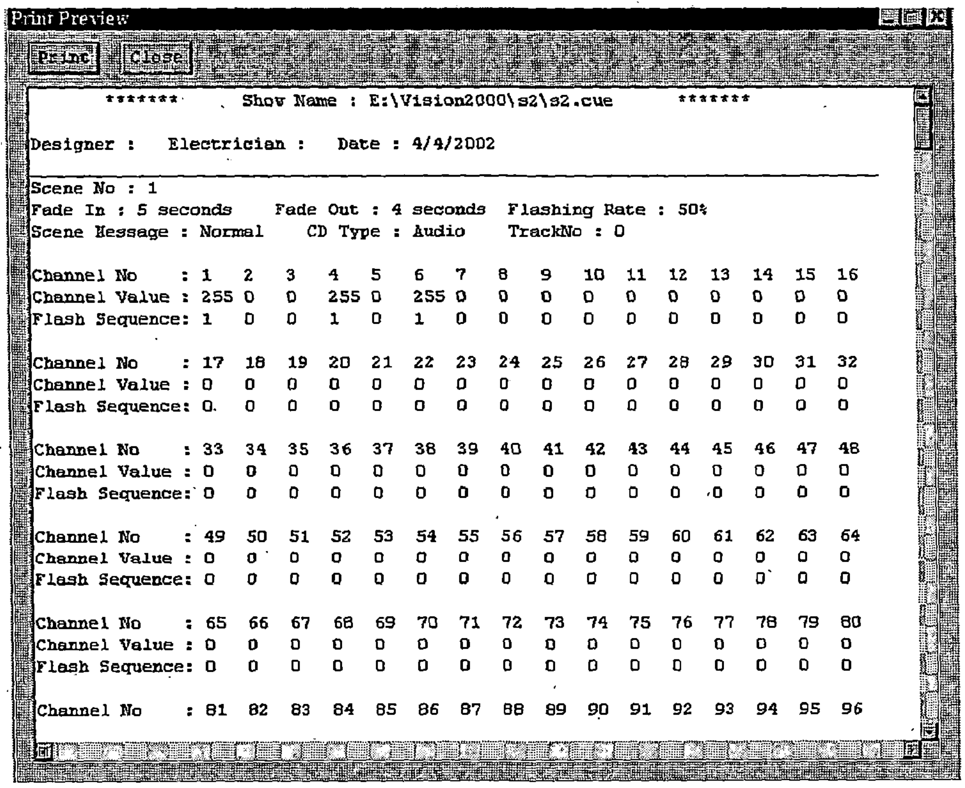
( ) Scene Range

From : 1 To : 1

Show : sur.cue Total Scene : 1

In print menu you can print the SceneSetting and the file in which all the information related to SceneSetting are stored.

Print Preview:



```
     *******     Show Name : E:\Vision2000\s2\s2.cue       *******

Designer :   Electrician :   Date : 4/4/2002
________________________________________________________________
Scene No : 1
Fade In : 5 seconds    Fade Out : 4 seconds  Flashing Rate : 50%
Scene Message : Normal    CD Type : Audio    TrackNo : 0

Channel No     : 1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
Channel Value : 255 0   0   255 0   255 0   0   0   0   0   0   0   0   0   0
Flash Sequence: 1   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0

Channel No     : 17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
Channel Value : 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Flash Sequence: 0.  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Channel No     : 33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
Channel Value : 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Flash Sequence: 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Channel No     : 49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
Channel Value : 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Flash Sequence: 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Channel No     : 65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
Channel Value : 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Flash Sequence: 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Channel No     : 81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
```

ReadMe

(a) Designing a Show

# You can design a new show by

* *Using the patch option from the menu bar*

On main window you can make up scenes one by one up to limit of hard disk.

<u>Start with scene 1:</u>

1. Enter the Designer name
2. Electrician Name
3. Date
4. Scene Number
5. Set Fade In time for Dimmer's Scenes (in seconds).
6. Set Fade Out time for the Dimmer's Scenes
7. Select Add/Cross option for Dimmer's Scenes

Note: First scene will ignore ADD/CROSS option

In case ADD is selected existing scene will not fade Out, instead all the changed Faders for next scene will be added as per their Fade In time. In case CROSS is selected, existing scene will Fade Out as per its fadeout time and the changed faders will come gradually as per their Fade In time.

8. Set the Flash /Chase speed (1 to 100%) for the faders (dimmers).

   You adjust scrollbar by clicking on up arrow or down arrow of it. This action set the value of channel into channel's edit box.

11. If you want, you can set any particular channel as Flash or Chase pattern (1-8). Using any option above will freeze the respective fader to Flash. You can also use only flash for any scene. The maximum chase sequences available are 1-8 so use may of these numbers to set the chase pattern for corresponding fader. To accomplish this task fast you can set the chase value in the chase window and then can type that value for all the chasers just by clicking in the chase Box.

13. Now your first scene is ready if you want to see the output just click Transmitter tool button. Of course you can use this option at any stage of designing.

14. Simply change the scene numbers by clicking on new button for new scene.

15. Save your show at times when the parameters are feeded or changed so that you may not loose the show data if power failure occurs.

<u>Insert Media in to a Show:</u>

Click on Tool Menu, it will open media dialog box.

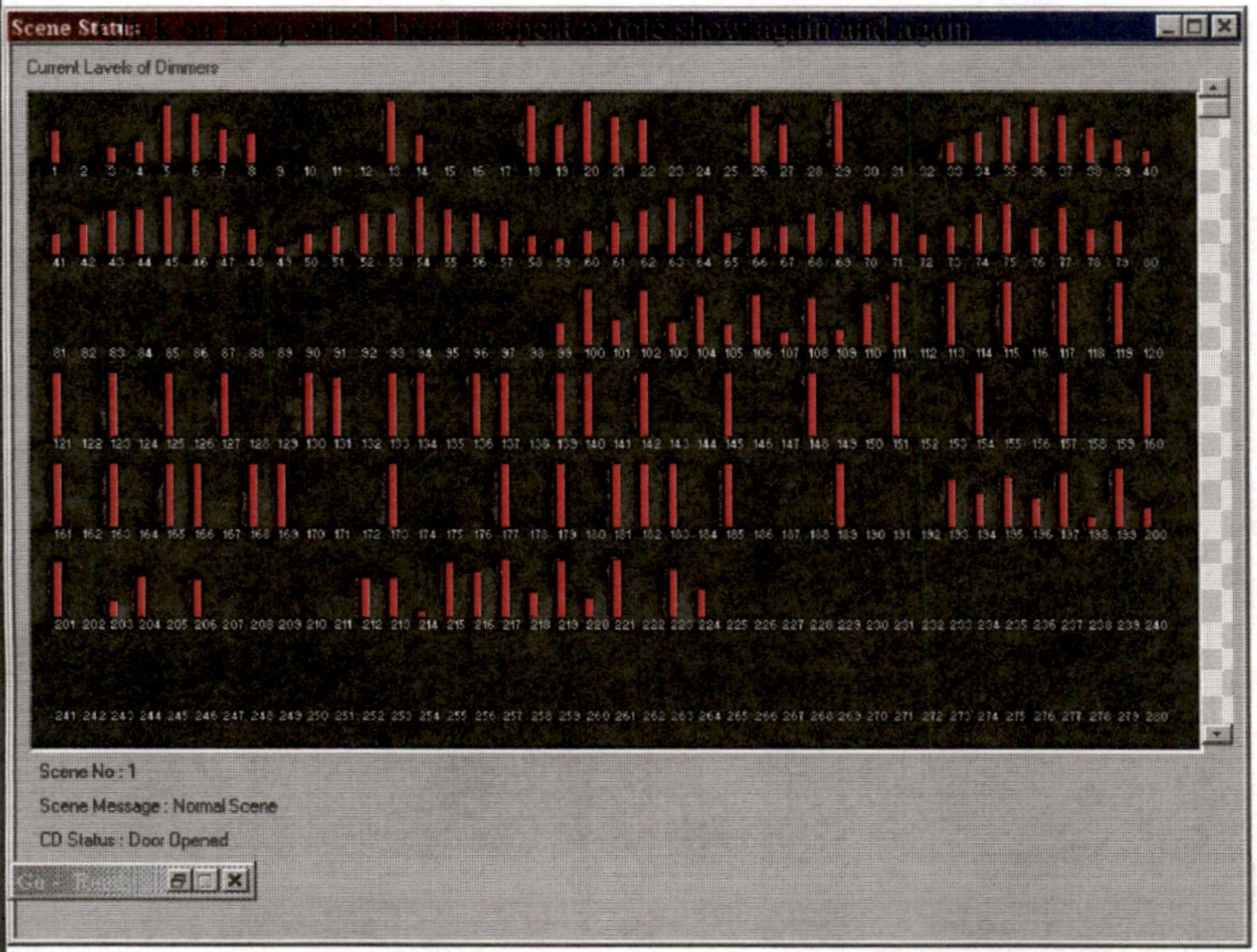
Click on Browse button to browse media file (mp3 audio or video file (AVI, DAT) or any other types).

Do same steps for more scenes.

- Watch scene status, Show Status and Audio/Video status in the windows provided for the purpose.

(b) AutoRunning a Show

- Once you have designed and recorded the show, to run it automatically, use 'Run' option from Auto Run menu. The status window is automatically open to display all live actions.



Scene No : 1

Scene Message : Normal Scene

CD Status : Door Opened

Now just relax and enjoy the show
while it runs fully automatically

Only registered users will receive latest software updates plus technical support by e-mail/phone/fax.

Theatre Vision

Do's and Don'ts

- You can set maximum up to '8' patterns of chase for any show.
- If you change Fade In time or Fade Out time of a scene directly and if it is more than the scene time recorded, you will have to re-record it for correct output.
- To keep same scene blank or to keep same gap in between the scenes just set Fade In time of the next scene as 1 second and then record it for the desired scene time.
- You can create a file by clicking the patch button on the standard toolbar. You may need to select a different directory or drive to reach the desired directory where you had stored the show files.

- You can open show files made by this software only.

References

1. User mannual visual C++ ver.6.00 Tata McGraw hill publication year 1996
2. User mannual visual C++ ver.6.00 Tata McGraw hill publication year 1998
3. User mannual visual C++ ver.6.00 Tata McGraw hill publication year 2000
4. Complete reference visual C++ ver.6.00writer Herbert shield
5. User mannual of DMX512, a user learning guide Artistic vision publication year 1990
6. User mannual of DMX512, a user learning guide Artistic vision publication year 1994
7. 21days visual C++ year of publication 2001