Shri G.S Institute of Technology & Science

Auto Text

Vaibhav Jain

Ravish Laad

Sanjay Singh

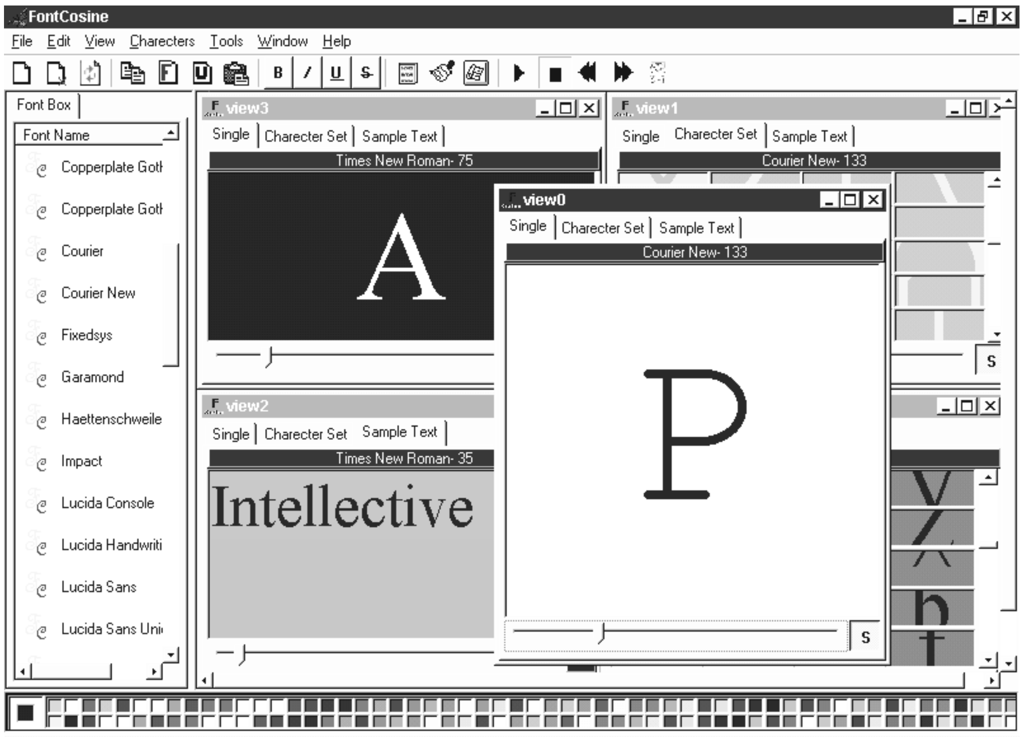
Auto-Text 1.0

Project Report

DEPARTMENT OF COMPUTER APPLICATIONS

- **Vaibhav Jain**
- **Ravish Laad**
- **Sanjay Singh**

FontCosine 1.0



What Do You Wanna Type Next?

Guided By:
Mr. Vishal Khasgiwala
Mr. Ashish Jain
Mr. Atul Jain

Developed By:
Vaibhav Jain,
Student, B.C.A V semester

Head Office	**Institute Campus**
182, Jaora Compound, **Indore.** Swati Jain Academy (SJA)	**33, Sampat Farms,** **Bicholi Mardana,** **Indore** Swati Jain Institute of Management Studies(SJIMS)

CERTIFICATE

This is to certify that **Mr. Ravish Laad(Roll-128), Mr. Vaibhav Jain(Roll-129) & Mr. Sanjay Singh(Roll-130)** *are the enrollee of Master of Computer Application course at* **Shri G.S Institute of Science & Technology** *and have together worked on the project* ***"Auto-Text 1.0"****. They have put sincere efforts in the project and have performed tasks related to the project in the Department Computer Lab of this institute. This project must be considered as a partial fulfillment of the* **M.C.A –I semester** *examinations that are conducted in affiliation with* **R.G.P.V, Bhopal.**

Date: November 25, 2004

Signatures:

RAM KUMAR SIR

EXTERNAL

Acknowledgments

No man is born complete and we are no exceptions. When the times were tensed and it seemed like we should put the whole bunch of code into the recycle bin all that could sustain me was the support of friends and elders alike. We were lucky enough to be surrounded by friends and elders who are helpful and supportive. Without their help Project Auto-Text would have probably being never completed.

We are also greatly thankful to the member faculty at S.G.S.I.T.S Indore as well as Taslim Madam, our honorable H.O.D who were there when their support and that wonderful sense of humor was badly and eagerly needed.

Special Thanks to Ram Sir who beared with our late project submissions and odd replies during the pre viva. We promise not to be late again. Thanks also goes Mradul Sir whose wonderful ideas and inspirations make for some very great challenges for us to face in our lives. We will never forget you sir.

Project Auto-Text was a one of the most complex and profound undertaking that we have ever taken in our life. For the first we were able to see how complex it is to either develop fully functional software and how difficult it is to get it properly designed to ensure maintainability and more importantly extendibility.

And, finally a word of gratitude to our Parents, Friends, Brother, Sisters & Pets who were always there with their support, encouragement & wit; even though we were acting like crazy at times.

November 2004.

Vaibhav Jain **Ravish Laad** **Sanjay Singh**

Table of Contents

1. Introduction **1**

Introduction 2

Screen Shots 3

Features & System Requirements 4

Installation & Usage 5

2. Engineering Paradigm **6**

Software Engineering Paradigm 7

The Software Process Model 9

Problem Statement & Requirement Analysis 12

Functional Specifications & System Goals 13

Software Tools 14

3. Architectural Design **15**

System Architecture 16

Code Structure 17

4. Component Level Design **18**

The Main Module 19

The Keyboard Interrupt Handler 20

The System Timer Interface 22

The Popup Window Module 24

The Mouse Handler Module 27

The Application Interrupt Handler 29

4. Miscellaneous **31**

Bibliography & References 32

Future Enhancements 33

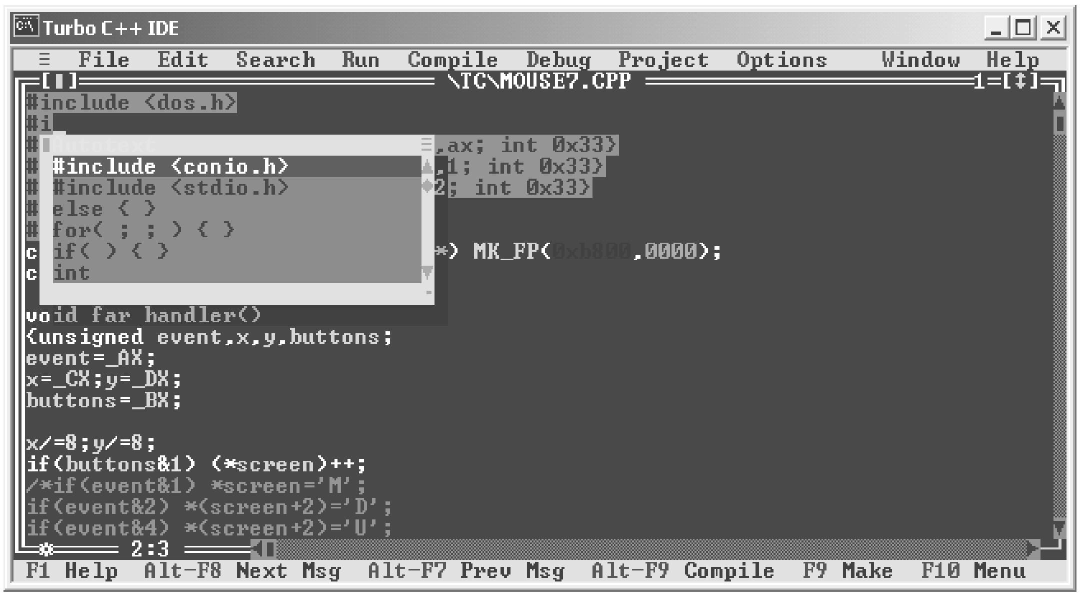
Inside the CD-ROM 34

Dedicated to Ever Green Friendship of Simba , Tbone and Pumba!!!

---VAIBHAV, RAVISH & SANJAY.

INTRODUCTION

Introduction



```
Turbo C++ IDE
 ≡  File  Edit  Search  Run  Compile  Debug  Project  Options   Window  Help
[▮]                         \TC\MOUSE7.CPP                          1=[↕]
#include <dos.h>
#i
# Autotext                    ≡,ax; int 0x33)
#  #include <conio.h>          ,1; int 0x33)
#  #include <stdio.h>          2; int 0x33)
#  else { }
#  for( ; ; ) { }
c  if( ) { }                    *) MK_FP(0xb800,0000);
c  int

void far handler()
{unsigned event,x,y,buttons;
event=_AX;
x=_CX;y=_DX;
buttons=_BX;

x/=8;y/=8;
if(buttons&1) (*screen)++;
/*if(event&1) *screen='M';
if(event&2) *(screen+2)='D';
if(event&4) *(screen+2)='U';
   2:3
F1 Help  Alt-F8 Next Msg  Alt-F7 Prev Msg  Alt-F9 Compile  F9 Make  F10 Menu
```

FIGURE 1

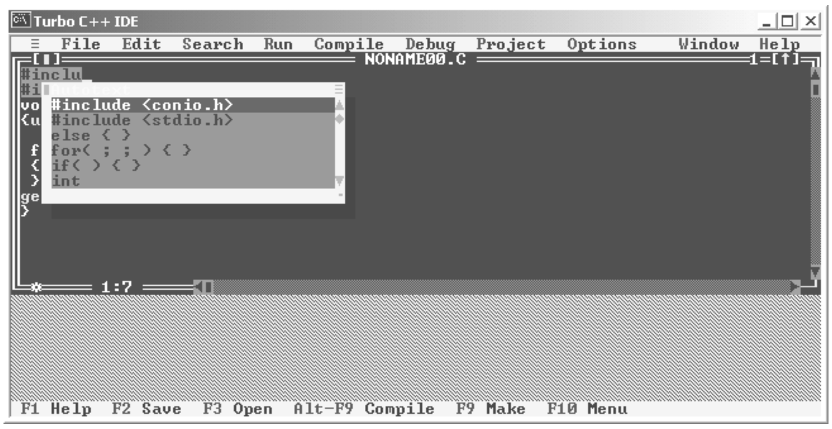
Auto-Text is a software to make life easier for people who usually type long and repetitive text with their keyboards. These people include programmers to frequently use a single text fragment throughout their long program. People to use a command line interface come next. In both such cases the speed of getting work done is roughly proportional to the speed with which text can typed from the keyboard. Now to improve one's typing speed he needs to practice with software such as Typing tutor etc. Obviously this is a long and cumbersome process. What we need is something that knows what do we want to type and then type it automatically for us. It should work much like the auto complete facility available to Visual Basic and Visual C++ programmers. To all such problems Auto-Text is your best answer.

Auto-Text gets it name from the AutoText feature that is available in MS-Word, since it tries to mimic its behavior. Like Word, this software captures the keystrokes that user is pressing and matches them with its predefine list of text phrases that have being provided by the user. If a suitable match is found then Auto-Text suggests it on the console screen via a popup window. If user wants to accept this suggestion then he can press the Enter key and WHEW!! the entire text is type for the user. As an example above picture depicts a 'C' programmer working in Turbo C++ 3.0 IDE. As he typed '#I' on its keyboard a popup window appeared on his screen that asking the user to complete it with phrase '#include <conio.h>'. The user then pressed the <Enter> key and the whole text was typed for him automatically.

However if the user is not interested in the suggested completion text then he can simply press the <Escape> key and Auto-Text popup will disappear won't bother him for the rest of the line. Transparency was our biggest goal and of major feature of Auto-Text during its development. Auto-Text is programmed to be as transparent to the user as possible and they can carry on their typing regardless of the fact whether Auto-Text has a completion suggestion or not.

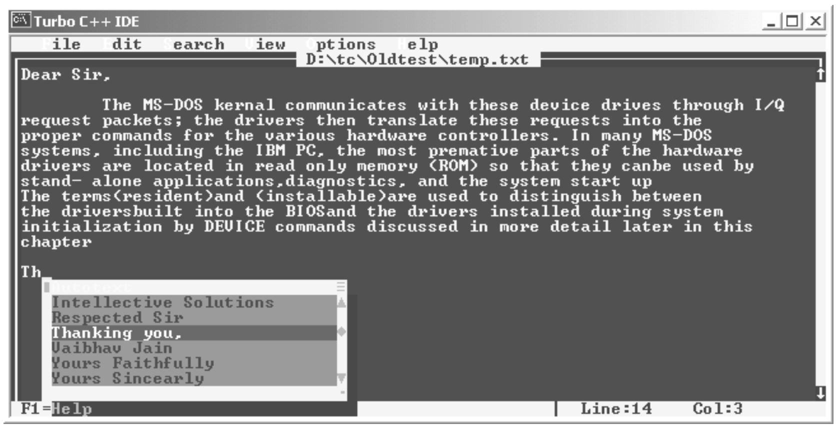
Screen Shots

Auto-Text popup window in Turbo C++ 3.0 IDE suggesting a completion text for inclusion of standard header file CONIO.H.


Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
NONAME00.C
1:7
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#inclu
#i
vo  #include <conio.h>
{u  #include <stdio.h>
    else { }
 f  for( ; ; ) { }
 {  if( ) { }
 }  int
ge
}
```

Auto-Text popup window in the standard edit tool for MS-DOS suggesting a completion text for inclusion of a salutation in a letter document.

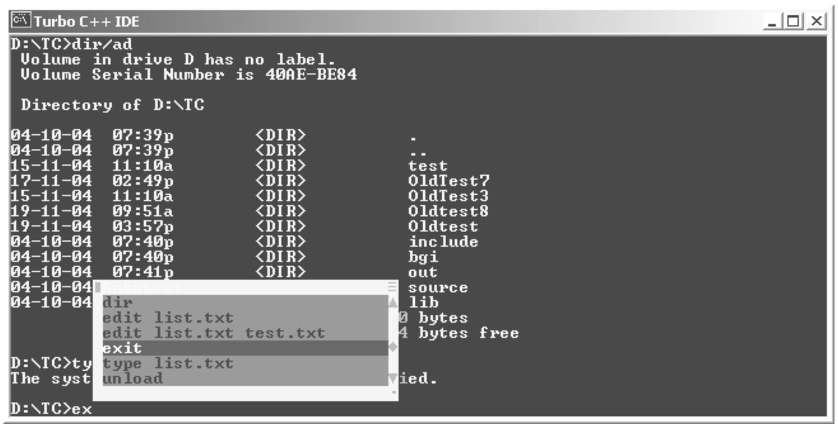


```
Dear Sir,

        The MS-DOS kernal communicates with these device drives through I/Q
request packets; the drivers then translate these requests into the
proper commands for the various hardware controllers. In many MS-DOS
systems, including the IBM PC, the most premative parts of the hardware
drivers are located in read only memory (ROM) so that they canbe used by
stand- alone applications,diagnostics, and the system start up
The terms(resident)and (installable)are used to distinguish between
the driversbuilt into the BIOSand the drivers installed during system
initialization by DEVICE commands discussed in more detail later in this
chapter

Th
   Intellective Solutions
   Respected Sir
   Thanking you,
   Vaibhav Jain
   Yours Faithfully
   Yours Sincearly
```

Auto-Text popup window at the command prompt of MS-DOS suggesting a completion text for a frequently used command.



```
D:\TC>dir/ad
 Volume in drive D has no label.
 Volume Serial Number is 40AE-BE84

 Directory of D:\TC

04-10-04  07:39p        <DIR>          .
04-10-04  07:39p        <DIR>          ..
15-11-04  11:10a        <DIR>          test
17-11-04  02:49p        <DIR>          OldTest7
15-11-04  11:10a        <DIR>          OldTest3
19-11-04  09:51a        <DIR>          Oldtest8
19-11-04  03:57p        <DIR>          Oldtest
04-10-04  07:40p        <DIR>          include
04-10-04  07:40p        <DIR>          bgi
04-10-04  07:41p        <DIR>          out
04-10-04                               source
04-10-04 dir                           lib
         edit list.txt                0 bytes
         edit list.txt test.txt       4 bytes free
         exit
D:\TC>ty type list.txt
The syst unload                       ied.

D:\TC>ex
```

Features and System Requirements

Types your text Automatically

Auto-Text automatically types the text for you. Just type in first few letters of your text and press Enter & Auto-Text will simply type the rest of the text for you automatically.

AutoComplete Any Time Anywhere

Auto-Text is not tied down to any specific software to work. You can invoke Auto-Text from any text editing application, dos command like so forth and can uses its great features to complete your typing work faster.

Speed, Stability & Reliability

Unlike other TSR's that hog down system resources and make system unstable Auto-Text has being thoroughly tested to ensure rock solid system stability and reliability of the system while in operation. Highly optimized code ensures that Auto-Text does not a use too much system memory or slows down the system too much.

Easy to Use

Working with Auto-Text is a breeze with no previous training required. Simply install it start working. Auto-Text interface is fully graphical where in user can institutively select or modify the completion text for them. A menu driven interface ensures proper selection of completion text without interfering with application program user is working.

Application Programming Interface

Auto-Text comes with a powerful application programming that lets other developers to customize the behavior of Auto-Text to suite their application needs. Right from the completion text to the colors of the display text in the popup window can be easily modified or queried with its API. This API set makes Auto-Text a truly special product in its class.

System Requirements

- Intel 80x86/Compatible Processor
- IBM PC-AT/Clone System
- MS-DOS or compatible Operating System (Windows 9x/ME /2000/XP)
- Two or Three button Mouse/Pointing device
- VGA Display Adapter
- 8 M.B of RAM (16 M.B highly recommended)
- 5 M.B of Hard Disk space.

Installation & Usage

Installation

Auto-Text can be installed by simply copying following files from the installation disk to a separate empty folder in the system hard drive:-

1. Load.exe
2. Unload.com
3. Enable.com
4. Disable.com
5. List.txt

Loading The TSR

In order to load the Auto-Text TSR into main memory the LOAD command must be executed. This command resides in the installation directory of Auto-Text. This command reads the List file from the current directory and load the TSR into the main memory. If however the TSR module is already present in the main memory then this command terminates with an Error message.

Unloading The TSR

In order to unload the Auto-Text TSR from the main memory the UNLOAD command must be executed. This command too resides in the installation directory of Auto-Text. This restores any modification to the system data that were done by the Load tool and frees up the memory that was held by the memory resident module. Please note that even if Auto-Text is not present in the main memory the Unload command completes without any error message.

Customizing the Suggestion List

The Suggestion List containing all the valid suggestions to the user is stored in an ASCII text file named as List.txt that is stored in the installation directory of Auto-Text. For example if Auto-Text is installed in directory C:\Autotext then the path to the List file is C:\Autotext\List.txt. Users can open this file in any text editor an can edit the list stored in it by adhering to the format of List file which is as follows:

- Each line contains a single valid suggestion.
- There must be no blank lines in the file.
- Each line must not start with a blank character i.e. spaces or tabs
- Each line can be at most 32 characters long at maximum.

After editing the List file the Auto-Text TSR module must be first unloaded and then reloaded into the memory. This can be done via the UNLOAD and the LOAD command line utilities that are provided.

SOFTWARE ENGINEERING PARADIGM

Software Engineering **Paradigm**

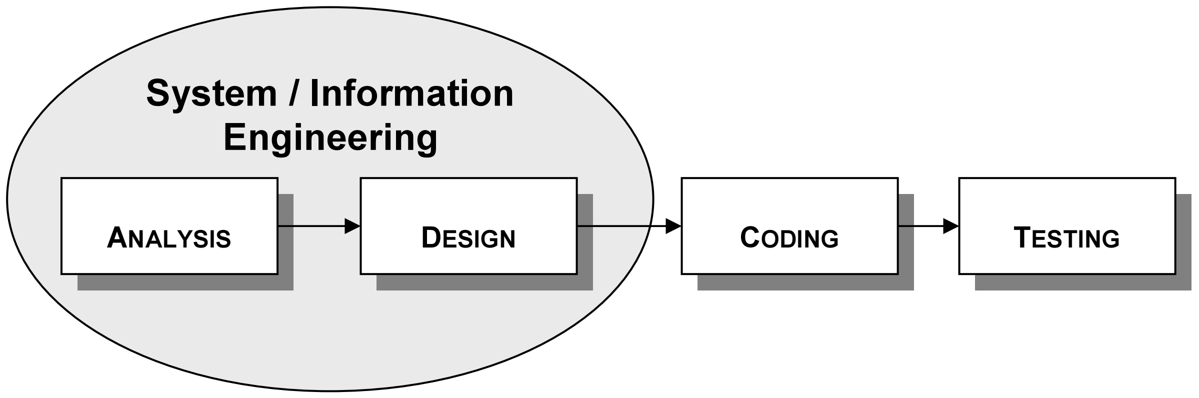


FIGURE 2

Software systems come and go through a series of passages that account for their inception, initial development, productive operation, upkeep, and retirement from one generation to another. This section explains the methods and model using which Auto-Text 1.0 was developed. It explains how Software Engineering Principals were applied to the development process and in selection of tools to produce the final product that meets the specifications as mentioned System requirements document. The section ends with the final design draft for coding the project.

Introduction

Software engineering is the computer science discipline concerned with developing large applications. Software engineering covers not only the technical aspects of building software systems, but also management issues, such as directing programming teams, scheduling, and budgeting. The Institute Of Electrical and Electronic Engineers -IEEE describes Software Engineering as:

> *"Software engineering. (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)"*
>
> \- **IEEE Std 610-1990.**

Today any person who is developing software for an individual or an organization must follow the principals of Software Engineering in order to produce software quickly and efficiently. Ignorance or negligence of these principles mostly results in project failure or software that does not meet the user expectations.

According to the principals of Software Engineering all Software Project should pass through some phases during their development cycle. This pursuance of a phases that act as a milestones to the project development process is known as he Software Life Cycle Model. The classic software life cycle models usually include some version or subset of the following activities:

Requirement Analysis

The requirement gathering process is intensified and focused specially on the software that is to be made. In this phase the Software Engineer (Analyst) tries to understand the nature of the prospective program. He outlines the information domain, functional requirements, behavior , performance and interface requirements that are expected of it. Requirements of the both the System and the Software are documented and reviewed with the customer.

Architectural Design

Completion of this phase requires definition of the interconnection and resource interfaces between system subsystems, components, and modules in ways suitable for their detailed design and overall configuration management. In essence design is a multi step process that focuses on dour distinct attributes of a program: Data Structure, Software Architecture, Interface Representations and Algorithm-tic details. It translates the system requirements into a representation for software that can assed for quality before coding.

Coding/Implementation

The Architectural design must translate into machine readable and understandable form that ultimately constitutes the destined software. This codifies the preceding specifications into operational source code implementations and validates their basic operation.

Software Integration & Testing

Passing this phase affirms and sustains the overall integrity of the software system architectural configuration through verifying the consistency and completeness of implemented modules, verifying the resource interfaces and interconnections against their specifications, and validating the performance of the system and subsystems against their requirements.

Maintenance

Software systems usually undergo change even after they are delivered. These changes include fixing of bugs discovered after delivery, adaptation required due to changes in external environments or functional enhancements that are required by the customers. In all such scenarios software is not redesigned or recreated instead existing systems are tweaked in order to meet these requirements. This never ending process is known are System Maintenance.

The Software Process Model

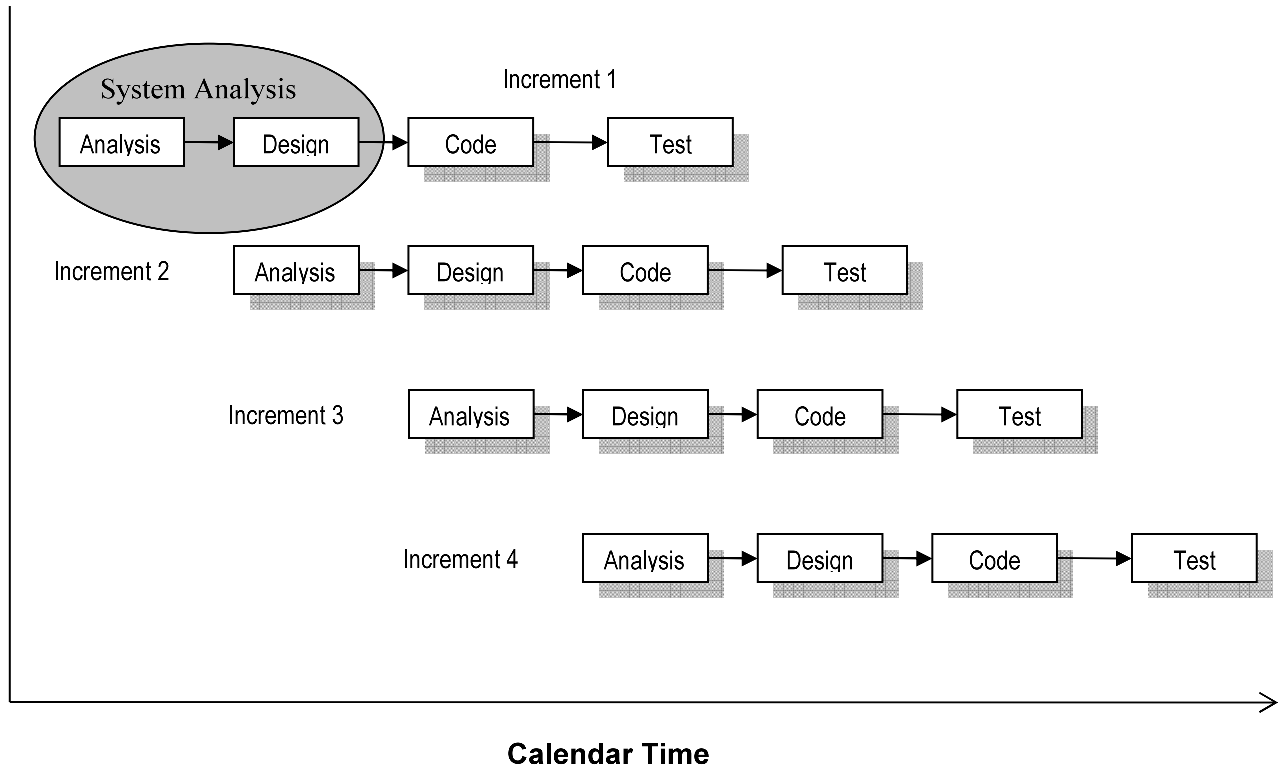


FIGURE 3

Software process models often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution. Such models can be used to develop more precise and formalized descriptions of software life cycle activities. Their power emerges from their utilization of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing. These model encapsulate all the necessary sub processes & tools to produce the final product.

The classic software life cycle as depicted in figure 2 is often represented as a simple prescriptive waterfall software phase model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order. Such models resemble finite state machine descriptions of software evolution. However, these models have been perhaps most useful in helping to structure, staff, and manage large software development projects in complex organizational settings, which was one of the primary purposes. Alternatively, these classic models have been widely characterized as both poor descriptive and prescriptive models of how software development "in-the-small" or "in-the-large" can or should occur.

However there is growing recognition that software like all complex systems , evolves over a period of time, Business and product requirements often change as development proceeds, making a straight path to an end product unrealistic; tight market deadlines make completion of a

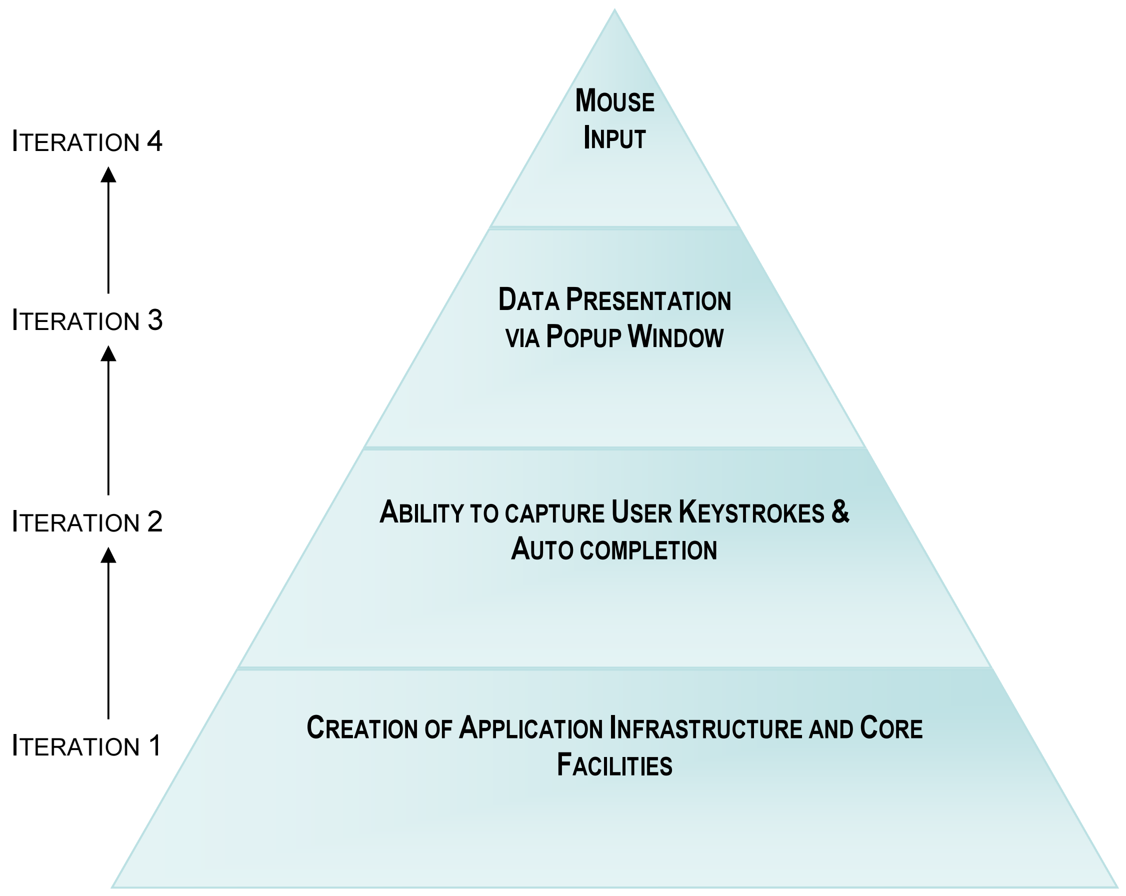

ITERATION 4
ITERATION 3
ITERATION 2
ITERATION 1
MOUSE INPUT
DATA PRESENTATION VIA POPUP WINDOW
ABILITY TO CAPTURE USER KEYSTROKES & AUTO COMPLETION
CREATION OF APPLICATION INFRASTRUCTURE AND CORE FACILITIES

FIGURE 4

comprehensive software product impossible, but a limited version must be introduce to meet competitive or business pressure, a set of core product or system requirement is well understood , but the details of product or system extensions have yet to be defined. In these and similar situations , software engineers need a process model that has been explicitly designed to accommodate a product that evolves over time. The linear sequential model in general is not designed to meet such demands. Thus what we need is a evolutionary process model for software development. Such model would be iterative and could be characterized in a manner that enables we as software engineers to develop increasingly complete versions of our software. To all the above problems the Incremental development Model proved to be our perfect solution such that it let Auto-Text evolve over time.

The Incremental Process Model

The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. The incremental model applies linear sequences in orderly fashion as the time progresses. Each sequence delivers a deliverable product. When an Incremental model is used the first increment is often a core product. That is , basic requirements are addressed but many subsequent features (know or unknown) are left undelivered. These core product is delivered and goes under a detailed review after which a plan for the next iteration is sketched out. This plan takes into consideration the changes requested to the previous version by the customer, and features that aren't yet addressed. The whole process in repeated in this way until the final product is delivered. It should be noted that each iteration can also follow the prototyping model. Any changes to the product specification made during the development process can easily be adapted to as compared to other process models. Thus incremental is a perfect model for projects having attributes to being dynamic as the final product usually achieves Higher Quality Rating because of entire customer requirements are being addressed in the final release.

In development of Auto-Text 1.0 the incremental model proved to be the perfect bet as because the development team was new to System Programming and the model made it possible for them to create the application as they learn it. Moreover since System programming is relatively error prone advancement to the next iteration was only done when all the previous iterations were tested successfully. Below each iteration that was done in the development process is described:-

Iteration 1

The first iteration in the development process of Auto-Text focused on creation of basic application facilities. This facilities included creation of requires of data structure and establishing of application architecture. The only functionality that was added in this iteration was the creation of completion text list that was read from the List file and stored into the main memory

Iteration 2

The second iteration mainly focused on the ability to capture user keystrokes into application buffer, to search into the completion text list to find the most proper match and to offer the user to complete it. If the user agreed than the application used to complete it text for the user.

Iteration 3

This iteration focused over the presentation and the user interface of the application. During this iteration the modules pertaining to the creation of popup window on system screen was implemented. Also the menu driven interface was realized with the help of few modifications to the Keyboard interface.

Iteration 4

In this last and final stage we implemented suitable support for mouse in the process of auto completion. The implementation made navigation with the help mouse possible. This feature together with menu driven interface did greatly improve the user friendliness of the application.

Since there was not a rigid time frame set for the project completion we were able freely producing product increments. Also the model made it possible to implements valuable suggestions provided by friends and colleagues. This model made let us achieve almost 90% of our design goals. However this accuracy in attaining the design goals could not be carried forward to the estimation of time taken for the project. The project required almost 200% more time of what was the estimated.

Problem Statement & System Requirements

The Problem

Users of DOS have being traditionally viewed up on keyboard friendly people. Although this accusation is not that much true still keyboard is a major and most widely used input device that is supported by MS-DOS. The support for mouse was not made available until version 3.0 and until now mouse are not natively supported by the MS-DOS. In addition there is a dearth of application that support mouse driven interfaces. Thus users only have the choice of using the keyboard for their work. However working with keyboard is quit slow and user productivity often depends on his ability to type faster. Thus any organization that invests in MS-DOS like command line / keyboard oriented interfaces must first invests in proper tool to train there employees to type faster. This solution is both unpractical and uneconomical. What we need is a tool that can predict what we want to write (just like our Mothers) and then type it for us without having us to do the same. What we need is such a system that can improve the productivity of users working with the keyboard without investing in costly training tools.

Requirement Analysis

✓ Since each user uses different and diverse application. So the system must not be tied to any particular product or vendor. It must be truly universal and must be able to adapt to any application / tool usage scenario.

✓ Since the system must run simultaneously with other application so it must ensure that it does not interfere with their proper functioning.

✓ Most probably the system will be implemented as an Operating System module and will probably interface hardware directly. In such a case the end system must be stable, robust and must be thoroughly tested for any errors or signs of instability.

✓ It must supplement as well as complement services that are provided by the text editing tools in choosing or viewing a particular font.

✓ System must provide an easy to use and intuitive interface to user so that no pervious training must be required in order to use the system.

✓ The system must be transparent the user so that in does not interfere in the way users uses the keyboard with out regard to the system itself.

✓ The project should be well planned and should follow most of the formal methods of Software Engineering with a proper process model. It should be completed within 1 month of time.

Functional Specifications & Design Goals

The platform for development will be MS-DOS as because it is the most widely used keyboard oriented Operating systems around. Also MS-DOS permits direct access to the system hardware which will be big asset to the project development process. Since MS-DOS only runs on Intel 80x86 or compatible platform we have no choice in choosing processor type.

Since the requirement analysis indicates that application must be integrate itself within any application he/she uses, we must implement the application as a TSR module that always remains the memory and watches user keystrokes for any completion hints.

The application must be coded in both a High level language and Assembly language. Coding in Assembly language will permit direct interaction with the system hardware and purposefully ensure fine-tuning of critical application procedures. The High level language will be used the implement the basic application infrastructure and foundation for the entire application. The HLL must be such that permits direct integration of assembly language with its code. The 'C' language was thus selected because this capability.

The application must capture the keystrokes that are made over the keyboard and must use then to suggest the a proper completion text to the user via a popup window that shows the suggested completion text with all the other completion texts so that users can select text other then the currently selected one.

The user partial text must be completed only if he accepts the suggestion. A mechanism must be ensured so that user can reject the suggested completion text. Also the mechanism must provide the user with ability to override the suggested completion text with his own manually selected completion text.

In order to provide users with an easy to use interface the popup window must be menu based that can be navigated with keyboard arrow keys as well mouse. Also their must be a simple way for the user to edit existing completion text data so that it can include only the texts that are in-fact used by the user frequently.

Since most text editing software's support mouse interface via which the user can take cursor to any screen position without using the keyboard. Such a behavior will be deterrent to Auto-Text as it will be able to complete the text accurately. In order to overcome this problem we need to capture any mouse events that are generated during the auto completion operation and handle it gracefully so as to ensure proper working of the software.

An application programming interface must be implemented through which other applications including third party application can so that they can customize the behavior of Auto-Text to suite their application needs. Right from the completion text to the colors of the display text in the popup window can be easily modified or queried with its API. This API set makes Auto-Text a truly special product in its class.

Software Tools

Auto-Text 1.0 is a 16-bit program, which runs on IBM-PC/AT compatible machines with Microsoft MS-DOS compatible operating system. Following tools were used to develop this version of the software.

Borland Turbo C++ 3.0

Borland Turbo C++ provides an integrated development environment where in user can write, compile & debug C/C++ programs. It includes comprehensive set of features that make writing C/C++ code a breeze. Even after more than 12 years of being released, it is the most widely used C/C++ development tool that is under use. Borland Turbo C++ was selected due to ease of use and most importantly its project management facilities.

Built-in Inline Assembler (BASM)

Turbo C++ include a built-in inline assembler (BASM) that makes it possible to include assembly language routines in C and C++ programs without any need for a separate assembler. Such assembly language routines are called inline assembly, because they are compiled right along with your C routines, rather than being assembled separately, then linked together with modules produced by the C compiler. This is big help to the programmers as they don't need to switch to and fro between two different tools and then to get them linked into a single unit.

Borland Turbo Assembler

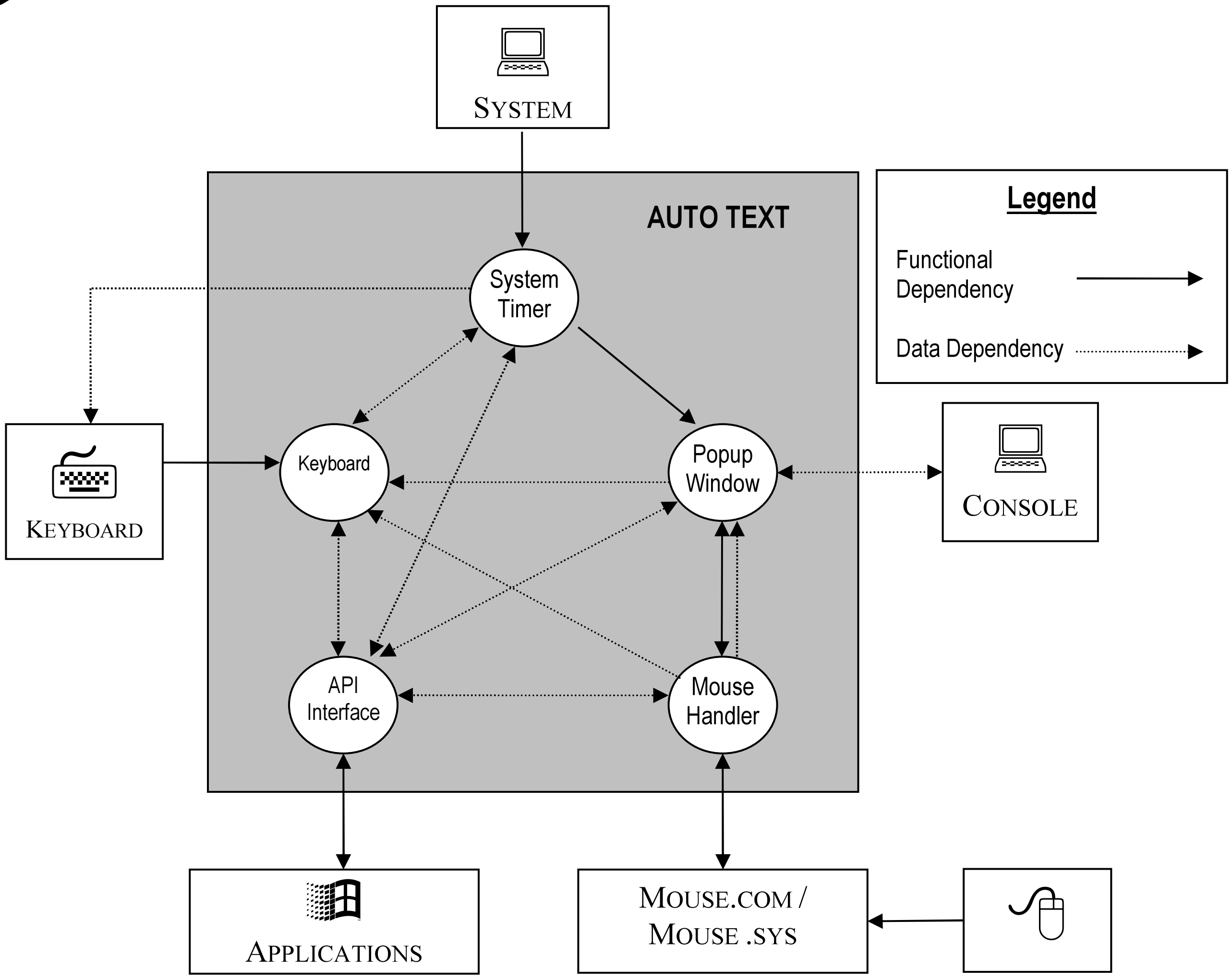
Borland Turbo Assembler(TASM) is a powerful and robust assembler from Borland International. It supports 80x86 instruction set and even supports 80286/80386/80486 protected mode processor instruction. Apart from standard features like macros and pseudo instructions such as WHILE, LOOP; TASM comes with Turbo Debugger that really does a good job in debugging system modules. A big advantage of TASM is that it can be directly integrated into Turbo C++ IDE from where it can be directly invoked without switching to command line. Turbo Assembler was selected due to ease of use and most importantly its project management facilities.

Microsoft Debug

'Debug' is a program testing , editing and debugging tool that's the part of almost all Microsoft operating systems right from the very first version of MS-DOS. Although still very quickie in terms of features when compared with monsters like Turbo Debugger; it provides all the basic facilities that are needed to debug any programs on the 80x86 platform. Debug although not feature rich still is invaluable dues to its ease of use , speed , accuracy and most importantly is significantly lower overhead. Debug was used in the development process to step into ISR module and Turbo C++ Integrated debugger is not capable of that. Moreover it was used to write small .COM programs such as UNLOAD, ENABLE etc.

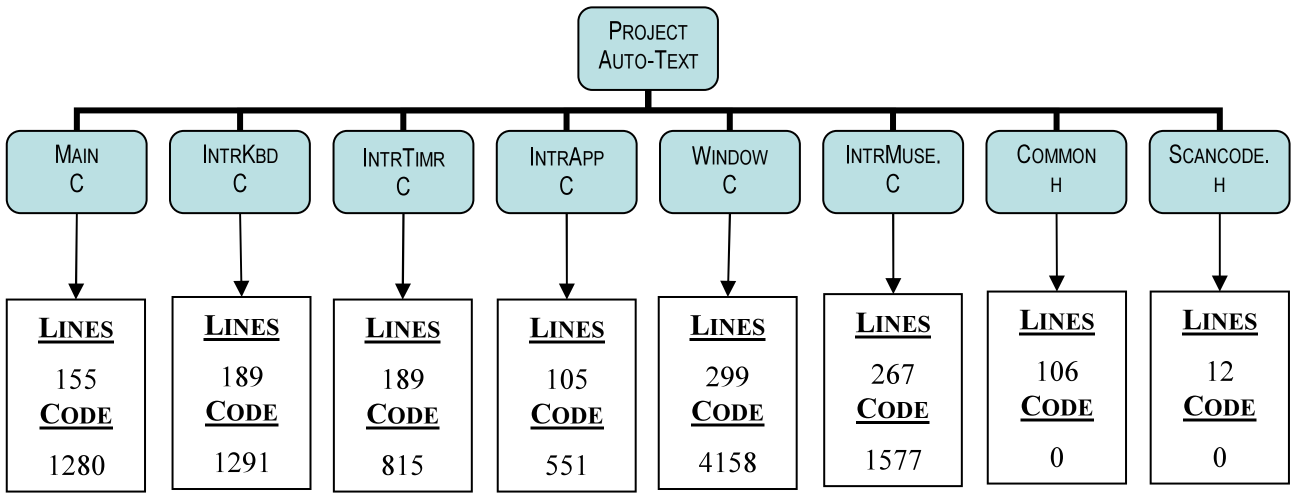
Architectural Design

System Architecture



Auto-Text was developed following an iterative development process model, so its architectural design was being constantly modified and fine-tuned at each iteration. Each of those modifications aimed improving application features, speed & stability so as to realize maximum out of the system specifications document. The end result of this process was a architecture that provided us with all these features. The design of Auto-Text called for partitioning of entire project into 4 distinct sections or module. Each module has a public interface through which other module communicate with it. Usually this interface consists of function but in order to minimize the use of stack a message passing client/server model of interfacing was used. In this model each module places a request for calling a method of other module. The Server ; In our case the Timer module; periodically scans its message queue for any pending requests and if one is found then is completes the request by calling the requested function on the requester's behalf. This setup ensures that usage of stack is minimized because there are no function calls. Instead the function calls are performed asynchronously by the system timer. Also before calling any function the Timer module switches to its own stack. In the coming section we describe in detail each and every module of Auto-Text, so as to complete the picture about the working of Auto-Text.

Code Structure



MAIN.C

This C Source file includes the Main module that is the entry point of the whole application.

INTRKBD.C

This C Source file implements the Keyboard Interface module that captures user's keystrokes and evaluates a valid completion text for it.

INTRTIMR.C

This C Source implements System Timer Interface module that accepts messages from all other module and completes them asynchronously.

INTRAPP.C

This C Source file implements the Application Programming Interface via which external application can query as well as modify the current state of Auto-Text memory resident module.

WINDOW.C

This C Source implements the Window module containing functions and setup necessary to create and manage the on screen popup window showing the currently suggested completion text.

INTRMUSE.C

This C Source implements the Mouse Handling module that captures any mouse events and converts them to an appropriate navigational command for the popup window.

COMMON.H

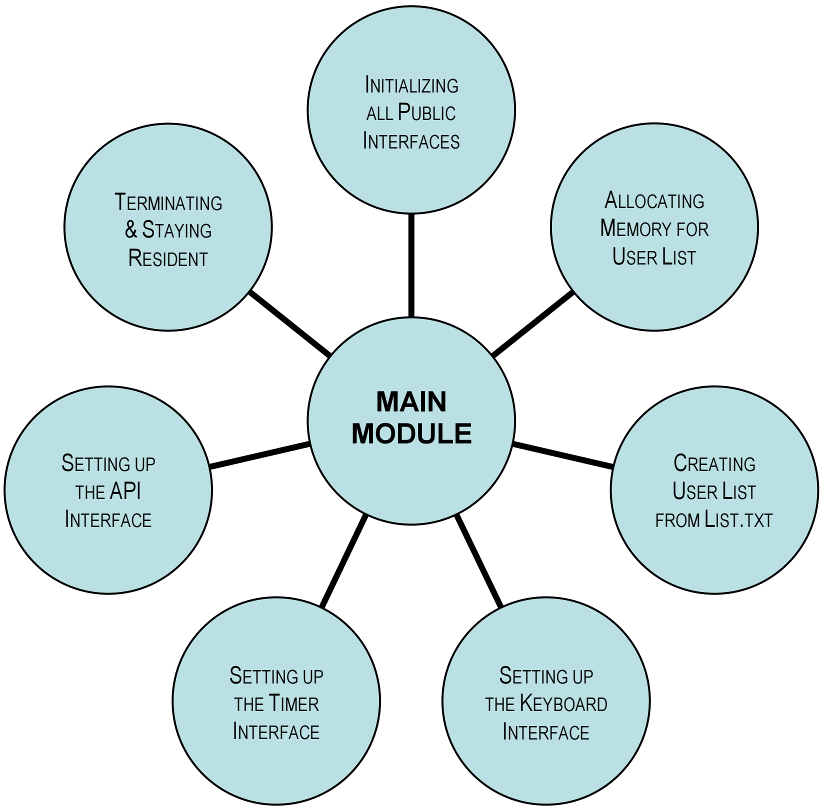
This C Header file includes all the public interfaces of all the modules. Each and every C Source file in the project includes this file.

SCANCODE.H

This C Header file includes scan code values various keyboard keys that are generated by the keyboard interrupt. This file is included in the IntrKbd.c file that implements the Keyboard Handling module.

COMPONENT LEVEL DESIGN

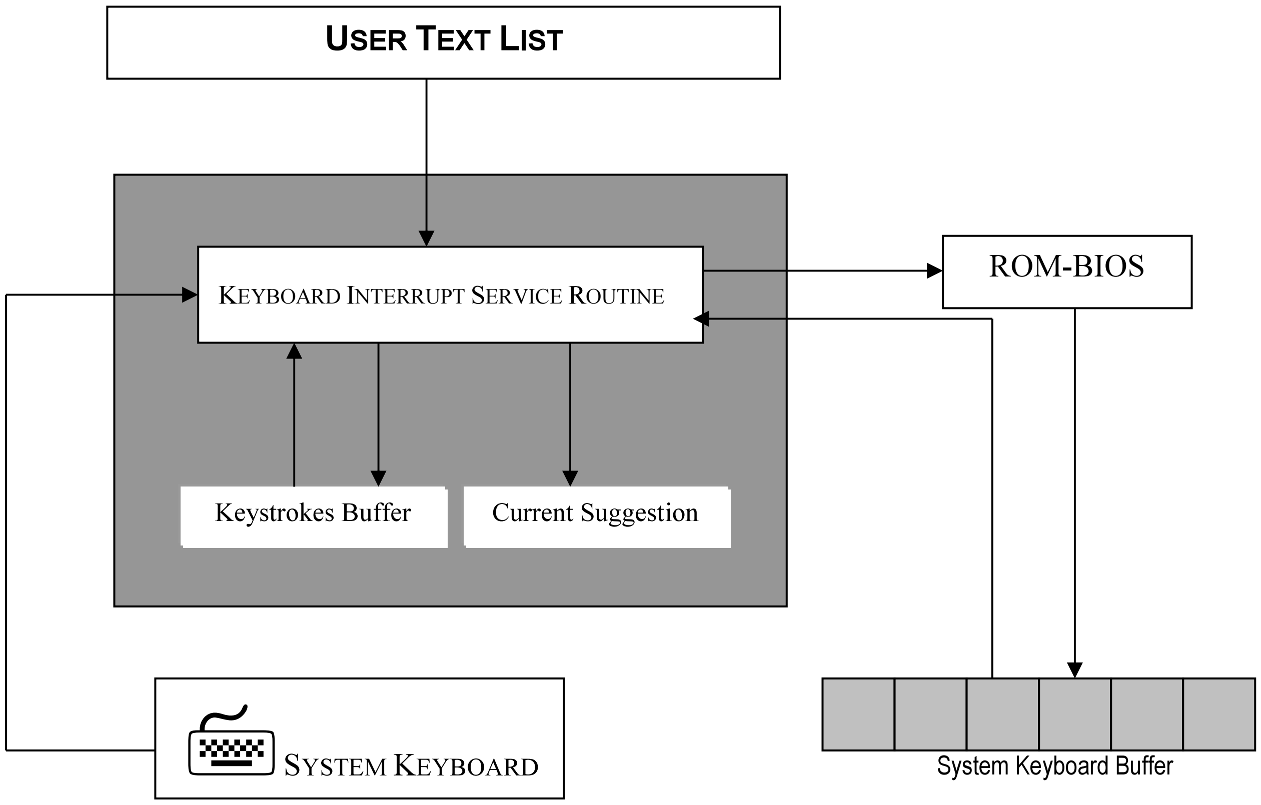
The Main Module


INITIALIZING ALL PUBLIC INTERFACES
TERMINATING & STAYING RESIDENT
ALLOCATING MEMORY FOR USER LIST
MAIN MODULE
SETTING UP THE API INTERFACE
CREATING USER LIST FROM LIST.TXT
SETTING UP THE TIMER INTERFACE
SETTING UP THE KEYBOARD INTERFACE

"The Journey of Thousand Miles Starts with the first single step" – Ancient Chinese saying

Like every other thing Auto-Text also starts with its first single step and that single step is the invoking of the Main module. The Main module that is defined in the file "MAIN.C" contains the function main() that serves as the entry point for the whole application in 'C' Language. The diagram above describes the functionality of the main module. The responsibilities of the main module include allocating and creating the User list text that is read in from the file List.txt which resides in the current directory. The second major responsibility of the main module is to initialization of the all the public interfaces that reside in the COMMON.H file. All the members of these interfaces are first nullified and then corresponding values in each member is loaded. After initializing the public interfaces the main modules proceeds with setting up the system interrupt vector table for Keyboard (09h) , Timer (1Ch) and API (2Dh) so that they points to the our Implemented Interrupt routines. This forces the system to call these handlers instead of default ROM-BIOS routines. For example whenever the user presses a key interrupt (09h) is generated and the system automatically jumps to the function INTRKBD() in the INTRKBD.C module , that implements the keyboard handling interface. The last and the most important job of the main module is to reserving to memory for program code in the main memory so that after terminating the program memory is not over written by MS-DOS. This is done by calculating the size of the running program and performing a Terminate and Stay resident call to the MS-DOS via INT 21h.

The Keyboard Interrupt Handler



Introduction

The keyboard interface is the most important module of the Auto-Text project. The primary purpose of this module is to capture user keystrokes in a special buffer and then to compare this buffer with the User Text List. If a suitable match is found than it is suggested to the user through a Popup Window. However user must be able to reject this suggestion if he wants. Also applications usually have a menu structure that is driven by keyboard. In both the cases the Auto-Text must simply get out of the way. It is this requirement that was a challenge to implement and was ultimately achieved through a sophisticated algorithm.

Public Interfaces

The keyboard handler manages two different public interfaces. The first interface call the 'Keyboard' pertains to the working of the keyboard and capturing of the user keystrokes. The second interface known as the 'List' interface contains is used to manages the auto complete process. This data structure contains all the user text that were read from the List file. The keyboard module matches this current set of stored keystrokes with this list and it a match is found then informs the user by placing a request for activating Popup Window.

```
struct Keyboard
      {char enabled, buffer[MAX_BUFF], ibuff, waitforspace;
          unsigned char minchars;
      };
struct List
      {int count, icursel;
          char far *cursel;
          char far *list;
      };
```

ENABLED: Resetting this flag disables the capturing of user keystrokes by the Keyboard ISR. This flag can be modified with the help of API interface. When the keyboard ISR is disables it simply passes control to the ROM BIOS without processing the pressed key.

BUFFER: This buffer stores all the keys that were pressed by the user. This buffer is than matched with the User Text List for the best match. Any space character or a non-printable character resets this buffer.

MINCHARS: This special variable tells the Keyboard ISR to wait for 'n' number of characters to be inside the Keyboard Buffer before it starts matching it with the User Text List and start making suggestions via the Popup Window. This variable can also be modified via the API interface.

WAITFORSPACE FLAG: This special flag disables the Keyboard ISR until a space character such as Enter, Tab etc are pressed. This flag is set when user presses the Escape key when Auto-Text was suggesting a completion text in order to reject the suggestion. Setting this flag then resets the keyboard buffer and Auto-Text starts waiting for a space character. This flag can also be modified via the API Interface.

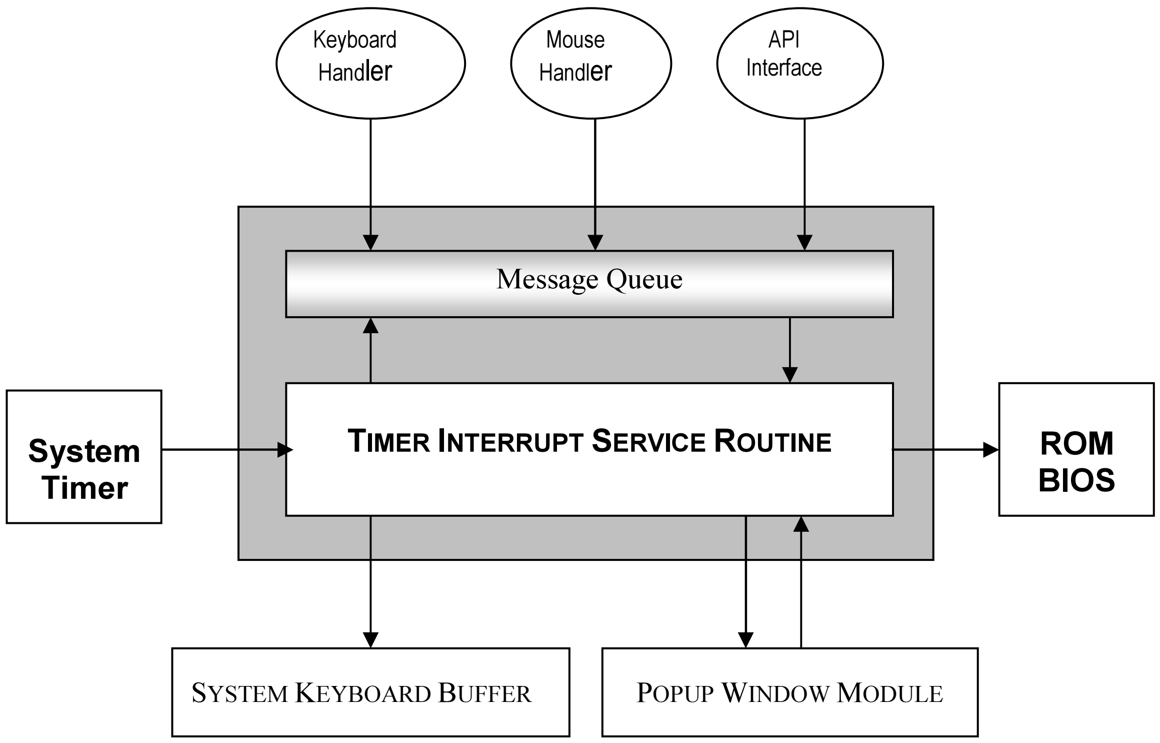
THE LIST ARRAY: This variable points to a memory array holds the user texts that were read from the List file during the load process. This array can be queried/modified via the API interface.

THE COUNT VARIABLE: This variable holds the number of user texts that are present in the array pointed to by the List variable.

THE ICURSEL VARIABLE: This element contains the index of the current suggestion from the List array. Its value is set by the Keyboard Module & used by all other modules to know about the currently suggested completion text.

THE CURSEL POINTER: This variable points to the ASCIIZ string of the currently suggested completion text. This pointer is then used by the Timer Module to complete suggested text for the user. The value of this variable is also available through the API interface.

The System Timer Interface



Introduction

Auto-Text is designed as a message passing system where is individual modules do not call procedure of other member modules. Instead they request a call a procedure by passing then a message via their public interface. This technique is used as because calling procedure requires the stack space of the for ground application and if Auto-Text goes down too much deep into the call hierarchy then it might end up overflowing the stack and thereby crashing up the whole system. So instead the procedure simply queues up a request for calling up the procedure. However since DOS is a single tasking system we cannot have multiple threads of control active at the same time as possible with Windows and other multi tasking operating system. So in order to complete the request the System Timer Interrupt is hooked. The System Timer generates an interrupt 18 times every second that is known as clock tick. This tick is used by the operating systems to update their clocks. So by hooking on the clock tick our Timer ISR gets called first switches to its own stack and then fetches a request from is message queue , does appropriate operation necessary to complete the request

Public Interface

```
struct Timer
    {   char enabled;           //enable or disable the Timer ISR
        char show_window;       //set if want to show window;
        char hide_window;       //set if want to hide window;
        char auto_complete;   //set if want to push data into keyboard
        char update_window; //set if want to update window;
        char scroll_down;      //set if want to scroll window down;
        char scroll_up;          //set if want to scroll window up;
        void interrupt (far *oldtimer)();// the old keyboard ISR
        };
```

ENABLED: The enabled flag when reset deactivates the process of the request completion of the timer module. In this state the input requests simply accumulate in the message queue of the module and the Timer ISR simply calls the ROM BIOS.

SHOW WINDOW: This message causes the Timer interrupt to show up the PopUp Window on the screen. This is done by calling the ShowWindow() function implemented in the Window module.

HIDE WINDOW: This message causes the Timer interrupt to Hide the popup window from the user screen by calling the HideWindow() function of the Window module. This message us usually generated by the keyboard module which when detects the <Enter> on the keyboard starts the process of auto completion.

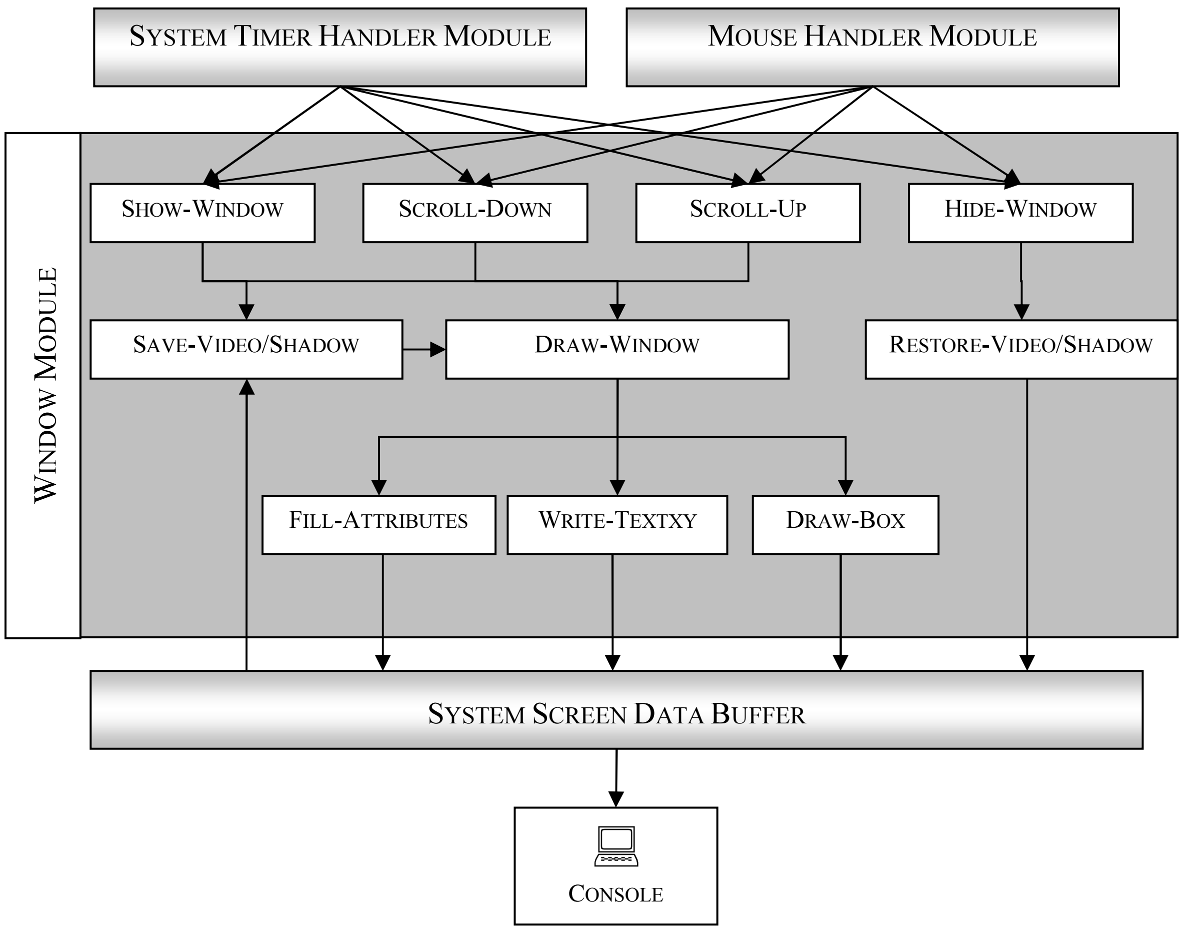
AUTO COMPLETE: This message is generated by the Keyboard module to request it to fill in the System Keyboard Buffer with some text. Since ISR must be very lean and should always complete quickly, so the keyboard module cannot cycle in a loop check weather the System Buffer is empty and then fill data into it. Instead it places a request with the timer, which then completes the request by placing some data from the completion text at every tick if the System Keyboard is found to be empty.

UPDATE WINDOW: This message is generated by the Keyboard and the Mouse modules in order to request redrawing of the Popup window on the screen. This message is generated by the use of navigational tools such as the arrow keys or the page-up/page-down keys that cause the window to scroll. The update message simply causes a call to the DrawWindow() function that only redraws the window on the screen.

SCROLL UP/SCROLL DOWN: This message is generated in order to request the timer to scroll the Popup Window up by one line. This message is generated by the use of navigational tools such as the arrow keys or the page-up/page-down keys that cause the window to scroll. The update message simply causes a call to the ScrollUp() function that only redraws the window on the screen.

OLDTIMER : This variable is initialized by the Main Module so that it contains the address of the previous ISR of the Clock Tick interrupt before replacing it with the address of its own Timer ISR. While unload the this value of used to restore the previous entry for the Clock Tick Interrupt ISR

The Popup Window Module



Introduction

The Window module which is implemented in WINDOW.C looks up to the presentation of the data to the user. This module is the heart of the popup window that Auto-Text flashes on the screen when it finds a matching suggestion for user text. The Popup Window is designed in such a way that it is visually attractive and not irritate the user with its constant negation. The Window module was one of the biggest and most complex module in the whole project. This complexity was due to the fact that it does not uses any of the DOS or ROM-BIOS functions to write on to the screen. Instead every this is written directly on to the screen memory. This ensures high speed and removes any hesitancy that might have arised while calling a DOS function in our TSR's. The diagram above describes the architecture of the Window Module.

Public Interface

```
struct Window
     {  char enabled;// show/not show the pop up window
          char active;// is the popup currently active or deactivated
          char row,col;//current location of popup window
```

```
        char height,width;
        char shadow;// wana show shadow?
        char trackcursor;//set if wana to track the cursor
        char maxheight,maxwidth;// minimum window size supported
        char minheight,minwidth;//maximum window size supported
        char bordercolor; // the border attribute for the border
        char fillcolor;// the fillin attribute for the window
        int uprange; // holds the upper most index to be show
        int dnrange; // holds the lower most index to be show
};
```

Module Methods

SHOWWINDOW()

This function queries the current location of cursor on the screen and then It function first saves the current screen data at the position that will be effected by the drawing of the window. Thirdly it initializes the Up range & Down Range variables so at to ensure proper scrolling and finally it calls the DrawWindow() function to draw the Popup Window on the screen.

HIDEWINDOW()

This function simply sets the window active flag to zero and restores the previous event handler for the mouse events. Finally is call the Restore Video function to restore the screen data back.

SAVEVIDEO()

This utility function saves the screen contents of the given rectangle in a provided buffer. This function is called by the showwindow() function to save screen contents before drawing the window on the screen.

RESTOREVIDEO()

This utility function restores the screen contents of the given rectangle that were stores in a given buffer. This function is called by the hidewindow() function to restore screen contents of the window rectangle to what were before drawing of the window on the screen.

SAVESHADOWATTR()

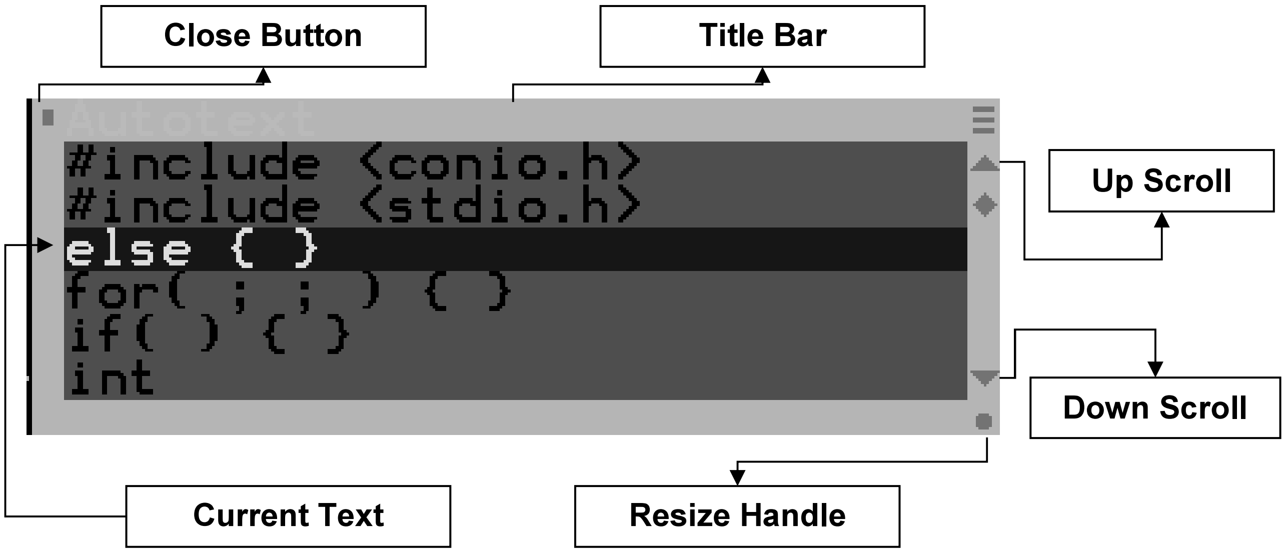
This function saves the character attribute data that falls into the shadow of the Popup Window into a temporary buffer.

RESTORESHADOWATTR()

This function restores the character attribute data that were overwritten by the shadow of the Popup Window from a temporary buffer.

DRAWWINDOW()

This function does the primary job of drawing on the screen. It uses the Up range and the Down Range variables to assess the correct set of data that is to be shown and then is draws the



```
#include <conio.h>
#include <stdio.h>
else { }
for( ; ; ) { }
if( ) { }
int
```

Window as shown in the diagram above by directly writing on to the screen. This function does not saves screen contents before drawing so it must be called only to update the window on the screen.

FILLATTRIBUTES()

This utility function fills the given rectangle of the screen with given character attributes. The new character attributes are specified with the help of an AND mask and a XOR mask. The attribute data of the given rectangle is first ANDed with And mask and then it XORed with the XOR mask. This function was used to fill the background color into the window.

WRITETEXTXY()

This utility function writes the given ASCIIZ string on to the given position in the console be writing directly to the screen memory. A maximum width parameter is also passed that tells this function , maximum number of character to write and if the width of the string exceed this number then the string is truncated and suffixed with periods.

DRAWBOX()

This utility draws a rectangle on the screen. The parameter passed includes the starting and ending rows, columns. If the rectangle exceeds the screen dimensions then the rectangle is automatically clipped.

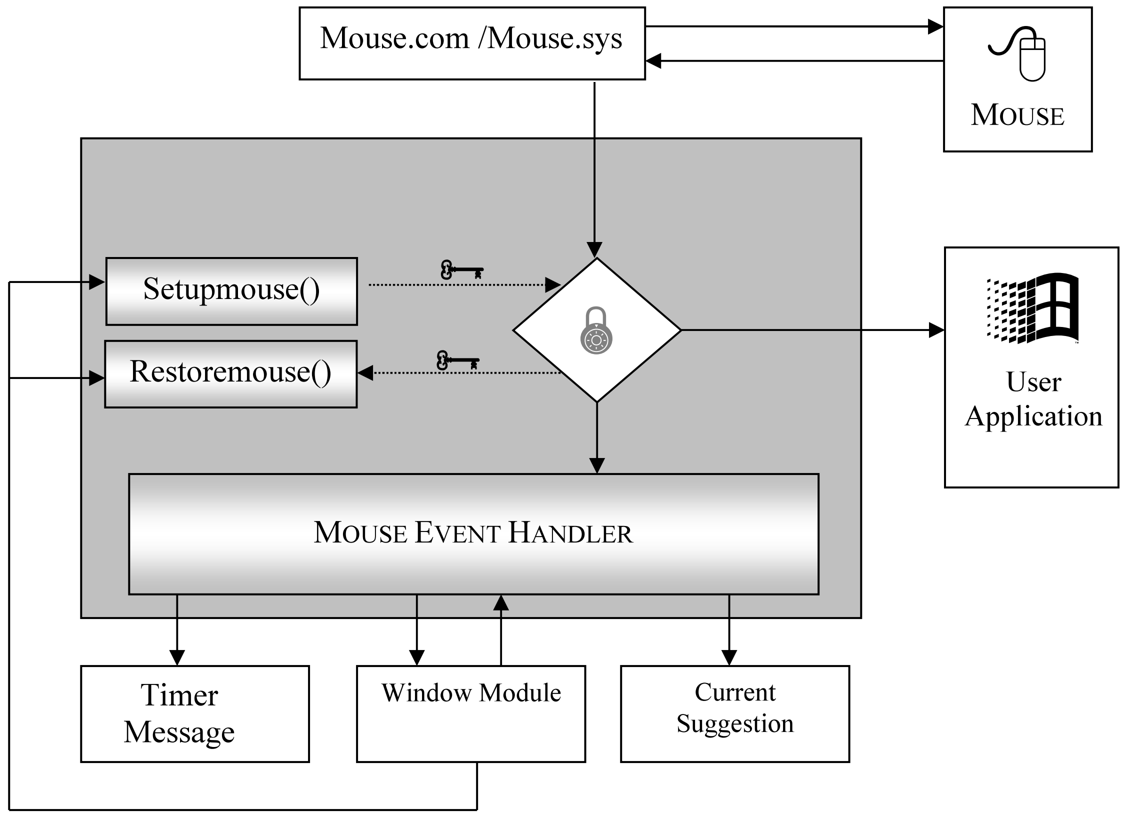
SCROLLUP()

This function simply decrements the current index of select text and redraws the window.

SCROLLDOWN()

This function simply decrements the current index of select text and redraws the window.

The Mouse Handler Module



Introduction

Ease of Use and user friendliness was one of the major specification and design goal of Auto-Text. To achieve this goal our team went upon to the difficult task of implementing a menu driven interface. In order to fulfill this goal further our team decided to make this menu mouse driven apart from being keyboard driven. Being mouse driven has given our application a distinct flexibility as using it we were able to implementing features such as Dragging and resizing of Popup Window. Using the drag feature the user can drag the Popup Window to any location on the computer screen. This function overrides the behavior of the popup window to track the location of the cursor and popping up around it. The resize feature lets the user resize the Popup Window to any preferred size. Both the facilities are very useful and have obvious advantages.

Module Members

SETUPMOUSE():

Since the mouse when active is in the control of the foreground application, so Auto-Text cannot simply seize the control of mouse forever. Instead the mouse is taken control of only when the Popup Window activates. The ShowWindow() function implemented in the Window module call this member function of Mouse module to set up the mouse so that Mouse.com call upon the event

DECISION TABLE FOR MOUSE EVENT HANDLER									
CONDITIONS									
Left Button Pressed	X			X	X			X	X
Left-Button Released			X				X		
Mouse at Title Bar	X								
Mouse at Scroll Bar				X					
Mouse at Size Bar					X				
Mouse at Close Button								X	
Mouse Moving		X				X			
Sizing Flag Set						X	X		
Dragging Flag Set		X	X						
ACTIONS									
Hide Window								X	X
Move Window	X	X							
Scroll Window				X					
Resize Window					X	X			
Set Sizing Flag					X	X			
Reset Sizing Flag							X		
Set dragging Flag	X	X							
Reset Dragging Flag			X						

handler Intrmouse() instead of the of calling the event handler of the foreground application. Also this function saves the address of old mouse event handler.

RESTOREMOUSE():

This function is called when the Popup Window is about and the control of the mouse is to be revoked back to the foreground application. This function reinstalls the old mouse event handler by calling upon the Mouse.com driver with its address. This address was saved by the Setupmouse() function when it was seizing the control of the mouse.

ISDRAGGING

This flag is set by the mouse event handler when it is the process of dragging the Popup Window around the console screen. This Popup Window is dragged by pressing the left mouse over the its title bar and then while keeping it pressed moving the mouse pointer to the desired screen location.

ISSIZING

This flag is set by the mouse event handler when it is the process of resizing the Popup Window. The resizing is done by dragging the lower right corner of the Popup Window with the help of the mouse.

The Application Interrupt Handler

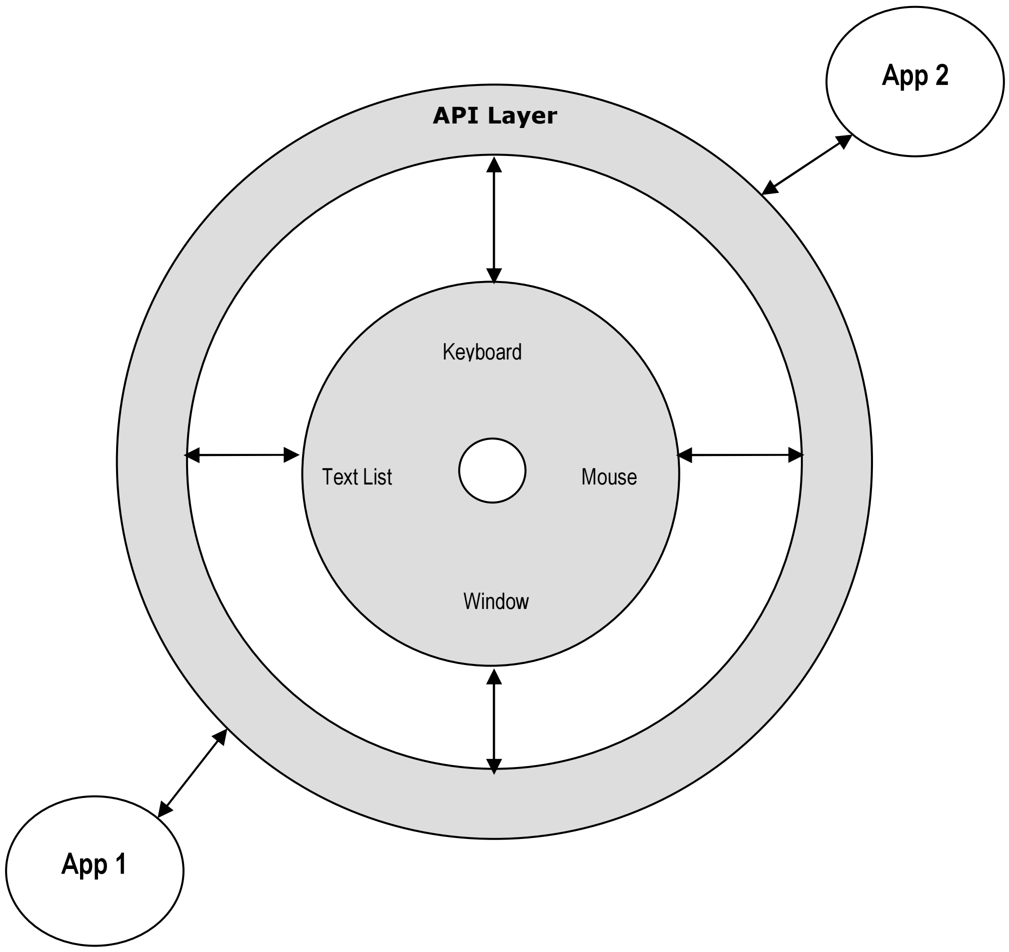


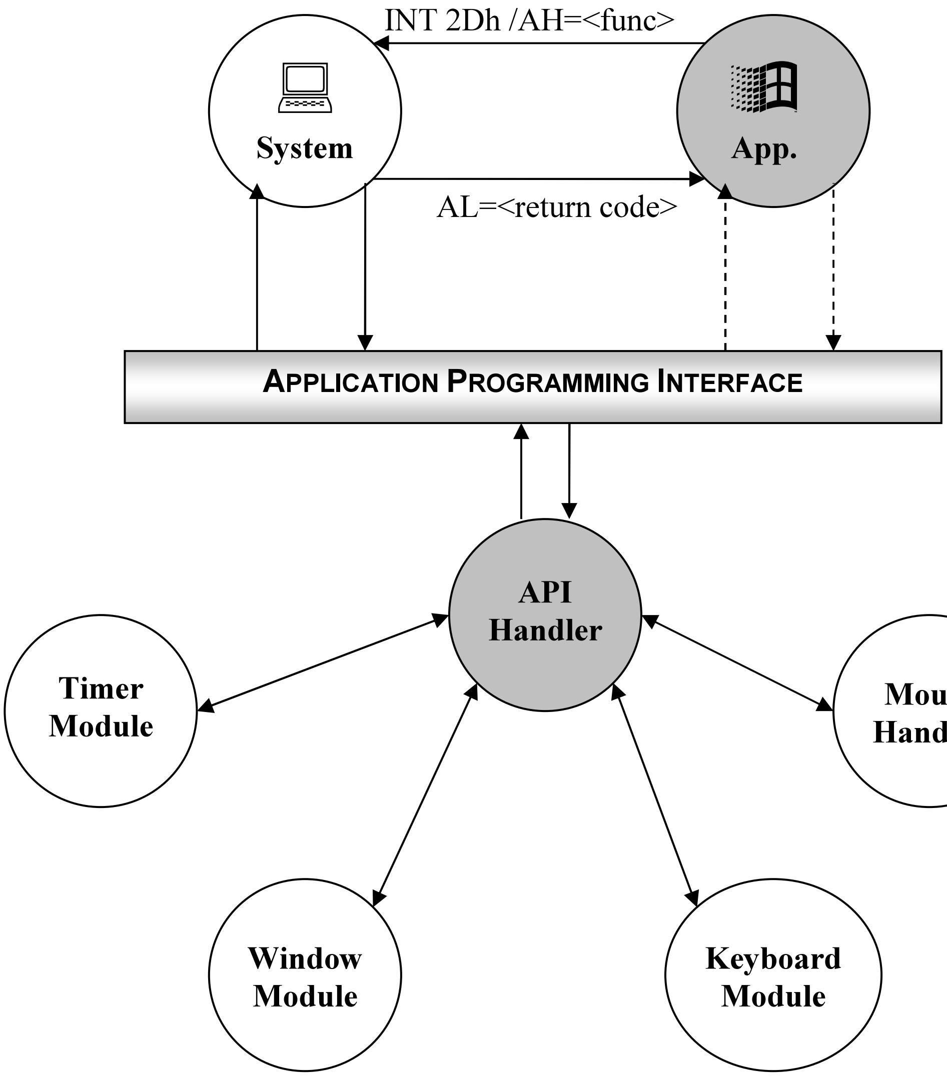
FIGURE 5

Introduction

The application interrupt handler module (INTRAPP.C) implements an the Application Programming Interface (API) for the Auto-Text module. The Auto-Text API provides the other application software a technique using which they can query information/data about current state of memory module or they can request some operation from it. For example an application can request Auto-Text to disable its popup window or it can ask Auto-Text to unload itself from memory. The API provides a comprehensive set of request through which any third part applications can modify the behavior of Auto-Text to suit it-self.

Module Level Specifications

Interrupt Service routine for INT-2Dh. This interrupt is known as a multiplex interrupt using which application programs can establish a communication channel between themselves. This interrupt is especially useful for TSR modules as because a TSR module can only respond to system interrupts and via this interrupt it can establish a communication protocol that can expose its



functionality to other programs. This is the only practical way using which messages/requests can be passed on the module. Below are the Specifications for the module implementing the Auto-Text API Interface: -

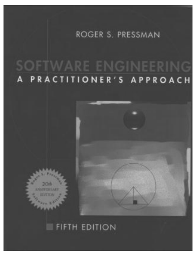
1. An ISR must be implemented that hooks to interrupt 2Dh.
2. On entry the ISR must check the value of AH register
3. If its not 6Eh then pass request the previous interrupt vector that was saved.
4. If its found to be 6Eh then must proceed with performing requested operation whose code is present in register AH
5. On completions of request the ISR must return a completion code in register AL that informs the application about whether the request was completed successfully or if an error had occurred.
6. If an request of unloading from memory is received than the ISR must restore all the previous interrupt vectors that were modified by the Main startup module, free all the memory that was allocated for data storage and lastly free up that memory that holds the code of the Auto-Text memory module.

MISCELLANEOUS

Bibliography & References

Software Engineering: A Practitioner's Approach

Rogger Pressman has written a solid comprehensive guidebook for the field of software engineering for both students of the discipline and software developers and managers practicing it. It's a classic textbook, clear and authoritive, with lots of pictures and examples.



Writer: Roger S. Pressman
Publisher: McGraw-Hill Publications.

Advance MS-DOS Programming

This book delves into the depths of MS-DOS architecture and addresses some key programming including device drivers, memory allocation, process management and wealth of other advance topics. Include a reference section detailing each and every MS-DOS function, error codes and version-by-version differences.

Writer: Ray Duncan
Publisher: B.P.B Publications, India

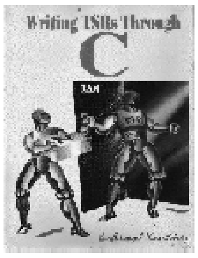
IBM PC Assembly Language And Programming

An excellent book for people who want to learn assembly language programming but finding it hard to grasp. In this Peter Abel in an intitutive way show his readers how to develop an assembly language program in a step by step manner.

Writer: Peter Abel
Publisher: Prentice Hall International.

TSR Programming in C

If TSR programming seems a black art to you, than this book is a must for you. This book slowly and methodically explains the readers about this incarnate art. Each and every technique to create your own TSR's with common misconceptions and pitfalls as thoroughly covered.



Writer: Yashwant Kanetkar
Publisher: B.P.B Publications, India

Future Enhancements

"To face tomorrow with the thought of using ideas and tools of yesterday is to envision life like at a standstill"

- James Bell

Off course improvement is a never ending road that leads to high quality and satisfaction. We believe that In order to create a product that achieves very high quality and customer satisfaction , the product needs to be continuously improved with more innovative ideas and features. In this context project Auto-Text is no exception. We indeed have ideas as to how to improve this product further. The enhancements below are just a glimpse of what we have in our minds:-

→ Providing a graphical tool to the user so that they can easily edit and manage their List texts, profile and can load or unload the memory resident module with it.

→ Ability to create profiles that can be activated in different applications. For example a profile for 'C" programming will contain 'C' language texts, another profile for DOS will contain frequently used dos commands.

→ Since currently Auto-Text can only run in the text modes, future version must be able to operate even in application running in graphical mode.

→ Automatic detection of frequently typed text and adding it up into the list. This will make possible for the user to add a new text without actually editing the list.

→ Create a similar utility for Windows platform.

Inside the CD-ROM

Auto-Text 1.0

Type without actually typing. Auto-Text 1.0 is headed your way. It automatically types the text for you. Be it lengthy statements of C language or long disclaimers at the end of your document. Auto-Text will type it for you at just a press of a button.

Location: \AutoText

Auto-Text Source Code

Full and complete source code of Auto-Text version 1.0. The code is fully commented and includes descriptive texts to understand it better. Needs Turbo C++ 3.0 compiler to compile which is also included.

Location: \Source

Auto-Text Project File

Full and complete documentation of the application source code and architecture of Auto-Text version 1.0. This file is also included in the accompanying CD-ROM to ready use template for your project files.

Location: \Doc

Turbo C++ 3.0

Borland Turbo C++ provides an integrated development environment where in user can write, compile & debug C/C++ programs. Include for compiling the source files.

Location: \Turboc

Ralph Brown Files

Ralph Brown Files are most comprehensive and exhaustive documentation of MS-DOS, ROM-BIOS , NOVEL NETWARE function calls, programming architecture up to date.

Location: \RBF

Send any comments or suggestions to vaibhavj02@yahoo.com