

**WAPLized Transmission of Voice Transformed Text Messages
Using
POP3 Protocol Implemented in .Net Framework**

Undergraduate Degree Project BESE-V

By

Capt Asif Saeed-ur-Rehman

Capt Afsar Hussain Qureshi (Project Leader)

Capt Niamat ullah Khan

Capt Bakhtyar Khan Durrani

Project supervised by

Mr. Nabeel Khan

*Dissertation to be presented as partial requirement for the award of
B.E Degree in Software Engineering*

**Military College of Signals
National University of Sciences and Technology, Rawalpindi**

We dedicate this humble effort to the

mother of

Captain Bakhtyar Khan Durrani

And

father of

Captain Asif Saeed-ur-Rehman

who breathed their last on 6th April 2002 and 23rd March 2003 respectively.

*May **ALLAH** rest their souls in eternal peace. (Ameen)*

ACKNOWLEDGEMENTS

We have no words at our command to express our deepest sense of gratitude to ALLAH ALMIGHTY who blessed us with knowledge, courage and strength to complete the project within stipulated time frame .

Rich tributes to our loving parents whose valuable prayers, salutary advices and emboldening attitude kept our spirits alive to strive for knowledge, which enabled us to reach this milestone.

We owe our deepest gratitude to our supervisor Mr Nabeel Khan for his valuable suggestions, positive criticism and prompt guidance. Without him it would have been almost impossible for us to accomplish this task successfully.

We extend our appreciations to all faculty members of the department for their cooperation and timely guidance during our studies in the university. Not mentioning the names of Dr Saeed Murtaza, Dr Nazre Haider, Brigadier S M S Tariq, Lieutenant Colonel Mofassir ul Haque and Major Wasique will be an injustice in this regard.

Last but not the least, we are grateful to our lady wives and adorable kids who remained patient and forthcoming, and seldom complained of our regular and long duration absences from sweet homes. Without their supportive attitude the dream of completing the project wouldn't have realized.

ABSTRACT

This document has been prepared as a dissertation of final year degree project to be presented to MCS/NUST in partial requirement for the award of BE degree in the discipline of Software Engineering.

The document discusses the transmission of voice transformed text messages. It includes the reasons for undertaking the project, the requirement analysis, the design and implementation phases. The aim of the project was to implement a semi-automated messaging system providing a variety of features. It also highlights the salient features of the technologies employed to complete the project.

The research and subsequent design and development of this project was carried out by Capt Asif, Capt Afsar, Capt Niamat and Capt Bakhtyar under the supervision of Mr.Nabeel Khan.

CONTENTS

DEDICATION	i
ACKNOWLEDGEMENTS	ii
ABSTRACT	iii
1. INTRODUCTION	1
1. Background	2
2. Operational Summary	2
3. Tools/Technologies Used	3
4. Features Available	3
5. Limitations/Boundaries	4
6. Future Enhancements	4
7. System Requirements	5
7.1 <i>Hardware Specifications</i>	5
7.2 <i>Software Specifications</i>	5
2. .NET FRAMEWORK	6
1. Introduction	7
2. Key Features of .NET Framework	8
2.1 <i>Common Language Runtime</i>	8
2.2 <i>Managed Execution Process</i>	10
2.2.1 <u><i>Designing and writing source code</i></u>	10
2.2.2 <i>Compiling code to MSIL</i>	11
2.2.3 <u><i>Compiling MSIL to native code</i></u>	11
2.2.4 <u><i>Executing the code</i></u>	12
2.3 <i>Common type system</i>	12
2.4 <i>.Net framework class library</i>	13
2.5 <i>Cross-language interoperability</i>	14
2.5.1 <i>Language interoperability overview</i>	14
2.5.2 <i>Common language specification</i>	16
2.6 <i>.Net framework security</i>	16
3. .Net framework in context	17
4. Automatic memory management	17

5. Allocating memory	18
6. Releasing memory	18
7. Overview of ADO.NET	19
3. POP3	21
1. Introduction	22
2. How does a POP3 mail server work?	22
3. Basic operation	23
4. States	24
4.1 <i>The AUTHORIZATION state</i>	24
4.2 <i>The TRANSACTION state</i>	25
4.3 <i>The UPDATE state</i>	25
5. Optional POP3 commands	25
6. Scaling and operational considerations	26
7. Algorithm for retrieving mail	27
8. Advantages of POP3 email	27
4. MESSAGING APPLICATION PROGRAMMING INTERFACE	29
1. Introduction	30
2. MAPI services and windows	30
2.1 Flexibility	31
2.1.1 <u><i>The tiered implementation of MAPI services</i></u>	<u>31</u>
2.2 Consistency	32
2.3 Portability	32
3. Messages	33
3.1 Text messages	33
3.2 Formatted documents and binary files	33
3.3 Control messages	34
4. MAPI applications	34
4.1 Electronic mail clients	35
4.2 Message aware applications	35
4.3 Message enabled applications	36

5. Other types of message applications	37
6. MAPI architecture	39
7. The MAPI client	40
7.1 <i>Messages and attachments</i>	40
7.1.1 <i>The message header</i>	41
7.1.2 <i>The Message body</i>	43
7.1.3 <i>Message attachments</i>	43
7.2 <i>Storage folders</i>	45
7.3 <i>Addresses</i>	46
8. The MAPI server	48
8.1 <i>Message transport</i>	49
8.2 <i>Message stores</i>	50
8.3 <i>Address books</i>	51
9. The MAPI spooler	51
5. TELEPHONY APPLICATION PROGRAMMING INTERFACE	54
1. Introduction	55
2. The telephony API model	55
2.1 <i>Lines</i>	56
2.2 <i>Phones</i>	57
3. TAPI and the WOSA model	57
4. Typical configurations	57
4.1 <i>Phone-based configurations</i>	58
4.2 <i>PC-based configurations</i>	59
4.3 <i>Shared or unified line configurations</i>	60

4.4	<i>Multi-line configurations</i>	61
5.	Telephone line services	62
5.1	<i>Plain Old Telephone Service (POTS)</i>	63
5.2	<i>Digital T1 lines</i>	63
5.3	<i>Integrated Services Digital Network (ISDN)</i>	63
5.4	<i>Private Branch Exchange (PBX)</i>	64
6.	TAPI architecture	64
6.1	<i>Assisted telephony services</i>	64
6.2	<i>Basic telephony services</i>	65
6.2.1	<i>The basic telephony line device API set</i>	65
6.3	<i>Supplemental telephony services</i>	66
6.3.1	<i>Supplemental telephony API for line devices</i>	66
6.3.2	<i>Supplemental telephony API for phone devices</i>	67
6.4	<i>Extended telephony services</i>	68
7.	TAPI hardware considerations	68
7.1	<i>Basic data modems</i>	68
7.2	<i>Voice-data modems</i>	68
7.3	<i>Telephony cards</i>	68
6.	SPEECH APPLICATION PROGRAMMING INTERFACE	70
1.	Introduction	71
2.	The model/architecture	71
2.1	<i>Automatic speech recognition</i>	72
2.1.1	<i>Shared</i>	73
2.1.2	<i>InProc recognizer</i>	74
2.2	<i>Text-to-Speech</i>	74
2.2.1	<i>Possible applications for Text-to-Speech</i>	75
3.	SAPI's strengths	75
4.	SAPI's Weaknesses	76
7.	WIRELESS ACCESS PROTOCOL	77
1.	Introduction	78
2.	Why WAP is necessary?	78
2.1	<i>Ensure interoperability</i>	78

2.2 Encourage and foster market development	79
2.2.1 The market is different	79
2.2.2 The network is different	81
2.2.3 The device is different	81
3. WAP specification	82
4. WAP solution benefits	84
4.1 Delivers an appropriate user experience model	84
4.2 Leverages proxy technology	85
4.3 Addresses the constraints of a wireless network	85
4.4 Provides a secure wireless connection	86
4.5 Optimized for handheld wireless devices	87
4.6 Implements new wireless functionality	87
4.7 Enables application development using existing tools	88
4.8 Adapts new standards for the industry	88
5. How developers benefit from using WAP-based solutions?	89
6. How subscribers benefit from using WAP-based solutions?	91

8. DESIGN AND IMPLEMENTATION 93

1. Use case diagram	94
2. Use case description	95
3. Sequence diagrams	98
4. Collaboration diagrams	103
5. Architecture design	108
6. Flow chart	109
7. Testing	111
8. Software testing methods (procedural)	111
8.1 Test case design	111
8.2 White-box testing	112
8.3 Basis path testing	112
8.4 Control structure testing	113
8.5 Black-box testing	114
9. Testing principles	114
10. Modular testing	114
10.1 Login	114
10.2 Main screen / Window	116
10.3 Administrator record	119
10.4 Customer record / Customer profile	123
10.5 Mail retrieval	127

		<i>10.6 Text to speech conversion</i>	<i>127</i>
9. REMOTE ACCESS	129		
		1. Introduction	130
		2. Links available	130
CONCLUSION	132		
ILLUSTRATIONS	133		
APPENDIX 'A'	151		
BIBLIOGRAPHY	158		

Chapter 1

INTRODUCTION

1. Background

Most of the technological development witnessed by today's common man is the logical outcome of the ongoing conflict between necessity and luxury. A scientific service which eluded the man of eighties as a dream has been transformed into reality and need for a man who has stepped into the 21st century. Internet and its allied facilities and applications is an example worth quoting in this regard which has confined the inhabitants of this world as members of a global village. Access to any sort of information in any form is no more a dream. Email has widely substituted the conventional means of message delivery in a time saving and cost effective way. Integration of mobile devices has added a new dimension to web and its applications. The user no longer wants to be dependent on a desktop PC, fixed telephone and an ISP to remain in touch with the latest happenings around the globe. The fact is more true for the persons whose nature of job demands constant travelling to remote areas lacking the above cited necessities together with a contradictory need to remain in contact with their enterprise and stake holders.

A need has been felt to develop an application for providing email access and sharing of information to the individuals living in remote areas devoid of basic Internet infrastructure. This very need forms the basis of our motivation to undertake this project, that too, employing the .NET FRAMEWORK, the technology of tomorrow. Talking in pure technical terms, the successful completion of our project has enabled the developed application to first go to the email server, download the email using POP3 algorithm and manage that email in to the database according to the user name and then make a call to that user to relay the mail in either voice or text format.

2. Operational Summary

The system is initialized with the administrator logging in. The administrator gets online through any ISP (Internet Service Provider) and establishes connection with the concerned POP3 (Post Office Protocol 3) compliant mail servers. The mails of the registered customers are retrieved through MAPI (Messaging Application Programming Interface) and stored in the local database, developed in SQL server 2000. The customer is then contacted through TAPI (Telephony Application Programming Interface). After validating the customer, his mails are transmitted in voice format through SAPI (Speech

Application Programming Interface), if the customer is subscribed to a line telephone. Whereas , in case the customer is subscribed to a mobile service, he has a choice to receive the mail either in voice or text format. For this, the application will employ WAP (Wireless Access Protocol).

3. Tools/Technologies Used

Following tools/technologies were employed to complete the project:-

- .Net Framework
- Post Office Protocol Version 3.0
- Messaging Application Programming Interface
- Speech Application Programming Interface
- Telephony Application Programming Interface
- Wireless Access Protocol
- SQL Server 2000
- ASP

4. Features Available

The salient features of the project are cited below:-

- Users can retrieve their mail in voice form on mobile phone, fixed phone.
- The application facilitates the following categories of individuals to retrieve their mail in voice form:-
 - Persons on move
 - Persons living in remote areas
- Administrator can view administrator's and customer's profile and can make necessary amendments.
- Administrator has the provision of carrying out all file operations like opening and saving a file supported by full fledged text editor.

- Administrator can compose and send the mail.
- Administrator can get print out of customer profile.
- The application is complimented by remote access feature. This has been done by developing and launching a website. The main purpose of the website is to allow users to update their profile especially their password. This ensures user's privacy which is of paramount importance in systems like this.

5. Limitations/Boundaries

Following are the boundaries/limitations of the project:-

- Application is limited to POP3 compliant mail servers only.
- The application supports only text to voice conversion and not the image and data conversion.
- Limited connectivity for mobile phone users.
- Application supports one client at a time.
- The application allows users to use only the default voice in TTS speaking mode.
- Application is semi-automated.

6. Future Enhancements

- With the use of multiple modems the application can support multiple customers simultaneously.
- TTS speaking mode can be further enriched to provide a choice to customers to select different voices.
- Application can be fully automated thereby keeping the role of administrator to the bare minimum.

7. System Requirements

7.1 Hardware Specifications

- Processor Minimum 600Mhz
- RAM Minimum 128MB
- Hard Disk Minimum 20 GB

7.2 Software Specifications

- Operating System Microsoft Windows 2000 Professional or Advanced Server 2000
- .NET Framework
- SQL Server 2000
- IIS
- Platform core SDK
- Speech SDK
- DirectX SDK

Chapter 2

.NET FRAMEWORK

1. Introduction

The .NET Framework is a new computing platform designed to simplify application development in the highly distributed environment of the Internet. The .NET Framework has two main components: the common language runtime and the .NET Framework class library.

The common language runtime is the foundation of the .NET Framework. We can think of the runtime as an agent that manages code at execution time, providing core services such as memory management and thread management, while also enforcing strict safety and accuracy of the code. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code; code that does not target the runtime is known as unmanaged code.

The .NET Framework class library is a comprehensive, object-oriented collection of reusable classes that we can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET and Web Services.

The .NET Framework also provides several runtime hosts, which are unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework SDK not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

Internet Explorer is an example of an unmanaged application that hosts the runtime (in the form of a MIME type extension). Using Internet Explorer to host the runtime enables us to embed managed components or Windows Forms controls (the .NET version of a Microsoft® ActiveX® control) in HTML documents.

2. Key Features of .NET Framework

Section	Description
Common Language Runtime	Explains what the common language runtime is, what it does, and the benefits it provides
Managed Execution Process	Describes what happens during managed execution
Common Type System	Identifies the types supported by the runtime.
Introduction to the .NET Framework Class Library	Introduces the library of types provided by the .NET Framework, which expedites and optimizes the development process and gives you access to system functionality.
Cross-Language Interoperability	Explains how we can ensure that managed objects written in different languages can interact with each other.
.NET Framework Security	Describes mechanisms for protecting resources and code from unauthorized code and users.

Table 1: **Key Features** of .NET Framework

2.1 Common Language Runtime

The common language runtime provides a code-execution environment that manages code targeting the .NET Framework. Code management can take the form of memory management, thread management, security management, code verification and compilation, and other system services.

Managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces security in a way that enables users to trust that although an executable attached to an e-mail can play an animation on screen or sing a song, it cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature-rich.

The runtime also enforces code robustness by implementing a strict type- and code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that managed code can consume other managed classes, types, and objects, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime ensures that the most common types of software issues are solved or eradicated completely. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management eliminates the two most common application errors, memory leaks and invalid memory references.

The runtime, coupled with the CTS, also accelerates developer productivity. For example, programmers can use their favorite development language, being absolutely assured that they can still take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, thus greatly easing the migration process for existing applications.

Although the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. A feature called **Just-In-Time (JIT)** compiling enables all managed code to run in the native machine language of the system on which it is executing.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Internet Information Services (IIS) and Microsoft® SQL Server. This enables us to use managed code to write our business logic, while still enjoying the superior performance of the industry's best enterprise servers.

2.2 Managed Execution Process

The managed execution process includes the following steps:

- ***Designing and writing source code***. To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.
- ***Compiling code to Microsoft intermediate language (MSIL)***. Compiling translates your source code into MSIL and generates the required metadata.
- ***Compiling MSIL to native code***. At execution time, a just-in-time (JIT) compiler translates the MSIL into native code.
- ***Executing the code***. The common language runtime provides the infrastructure that enables execution to take place as well as a variety of services that can be used during execution.

2.2.1 Designing and writing source code

To obtain the benefits provided by the common language runtime, we must use one or more language compilers that target the runtime, such as Visual Basic, C#, Visual C++, JScript, or one of many third-party compilers such as a Perl or COBOL compiler.

Because it is a multi-language execution environment, the runtime supports a wide variety of data types and language features. The language compiler we use determines which runtime features are available, and we design our code using those features. Our compiler, not the runtime, establishes the syntax our code must use. If our component must be completely usable by components written in other languages, our component's exported types must expose only language features that are included in the Common Language Specification (CLS).

2.2.2 Compiling code to Microsoft intermediate language (MSIL)

When compiling to managed code, the compiler translates our source code into **Microsoft intermediate language (MSIL)**, which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Before code can be executed, MSIL must be converted to CPU-specific code by a just in time (JIT) compiler. Because the runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and executed on any supported architecture.

When a compiler produces MSIL, it also produces metadata. Metadata describes the types in our code, including the definition of each type, the signatures of each type's members, the members that our code references, and other data that the runtime uses at execution time.

2.2.3 Compiling MSIL to native code

Before Microsoft intermediate language (MSIL) can be executed, it must be converted by a .NET Framework just-in-time (JIT) compiler to native code, which is CPU-specific code that runs on the same computer architecture as the JIT compiler.

Because the runtime supplies a JIT compiler for each supported CPU architecture, developers can write a set of MSIL that can be JIT-compiled and executed on computers with different architectures.

2.2.4 Executing the code

The common language runtime provides the infrastructure that enables managed execution to take place as well as a variety of services that can be used during execution. Before a method can be executed, it must be compiled to processor-specific code. Each method for which Microsoft intermediate language (MSIL) has been generated is JIT-compiled when it is called for the first time, then executed. The next time the method is executed, the existing JIT-compiled native code is executed. The process of JIT compiling and then executing the code is repeated until execution is complete.

During execution, managed code receives services such as automatic memory management, security, interoperability with unmanaged code, cross-language debugging support, and enhanced deployment and versioning support.

2.3 Common type system

The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:

- Establishes a framework that enables cross-language integration, type safety, and high performance code execution.
- Provides an object-oriented model that supports the complete implementation of many programming languages.

- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.

The common type system supports two general categories of types, each of which is further divided into subcategories:

- **Value types.** Instances of value types are stored as the representation of their value. Value types can be built-in (implemented by the runtime), user defined, or enumerations.
- **Reference types.** Instances of reference types are stored as a reference to the value's location. Reference types can be self-describing types, pointer types, or interface types. The type of a value can be determined from values of self-describing types. All self-describing types derive from a base type, [System.Object](#). Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

2.4 .Net framework *class library*

The .NET Framework includes classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, the .NET Framework types are CLS compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS).

The .NET Framework types are the foundation on which .NET applications, components, and controls are built. The .NET Framework includes types that perform the following functions:

- Represent base data types and exceptions.
- Encapsulate data structures.

- Perform I/O.
- Access information about loaded types.
- Invoke .NET Framework security checks.
- Provide data access, rich client-side GUI, and server-controlled, client-side GUI.

The .NET Framework provides a rich set of interfaces, as well as abstract and concrete (non-abstract) classes. We can use the concrete classes as is or, in many cases, derive our own classes from them. To use the functionality of an interface, we can either create a class that implements the interface or derive a class from one of the .NET Framework classes that implements the interface.

2.5 Cross-language interoperability

The common language runtime provides built-in support for language interoperability. However, this support does not guarantee that code we write can be used by developers using another programming language. To ensure that we can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

2.5.1 Language interoperability overview

Language interoperability is the ability of code to interact with code that is written using a different programming language. Language interoperability can help maximize code reuse and, therefore, improve the efficiency of the development process.

Because developers use a wide variety of tools and technologies, each of which might support different features and types, it has historically been difficult to ensure

language interoperability. However, language compilers and tools that target the common language runtime benefit from the runtime's built-in support for language interoperability.

The common language runtime provides the necessary foundation for language interoperability by specifying and enforcing a common type system and by providing metadata. Because all languages targeting the runtime follow the common type system rules for defining and using types, the usage of types is consistent across languages. Metadata enables language interoperability by defining a uniform mechanism for storing and retrieving information about types. Compilers store type information as metadata, and the common language runtime uses this information to provide services during execution; the runtime can manage the execution of multi-language applications because all type information is stored and retrieved in the same way, regardless of the language the code was written in.

Managed code benefits from the runtime's support for language interoperability in the following ways:

- Types can inherit implementation from other types, pass objects to another type's methods, and call methods defined on other types, regardless of the language the types are implemented in.
- Debuggers, profilers, or other tools are required to understand only one environment the Microsoft intermediate language (MSIL) and metadata for the common language runtime and they can support any programming language that targets the runtime.
- Exception handling is consistent across languages. Our code can throw an exception in one language and that exception can be caught and understood by an object written in another language.

To ensure that your managed code is accessible to developers using any programming language, the .NET Framework provides the Common Language Specification (CLS), which describes a fundamental set of language features and defines rules for how those features are used.

2.5.2 *Common language specification*

To fully interact with other objects regardless of the language they were implemented in, objects must expose to callers only those features that are common to all the languages they must interoperate with. For this reason, a set of language features has been defined, called the Common Language Specification (CLS), which includes basic language features needed by many applications. The CLS rules define a subset of the [common type system](#); that is, all the rules that apply to the common type system apply to the CLS, except where stricter rules are defined in the CLS. The CLS helps enhance and ensure language interoperability by defining a set of features that developers can rely on being available in a wide variety of languages. The CLS also establishes requirements for CLS compliance; these help us determine whether our managed code conforms to the CLS and to what extent a given tool supports the development of managed code that uses CLS features.

2.6 *.Net framework security*

The .NET Framework provides several mechanisms for protecting resources and code from unauthorized code and users:

- ***ASP.NET Web Application Security*** provides a way to control access to a site by comparing authenticated credentials (or representations of them) to Microsoft Windows NT file system permissions or to an XML file that lists authorized users, authorized roles, or authorized HTTP verbs.
- ***Code access security*** uses [permissions](#) to control the access, code has to protected resources and operations. It helps protect computer systems from malicious mobile code and provides a way to allow mobile code to run safely. (Code Access Security together with the policies that govern it is referred to as *evidence based security*.)

- **Role-based security** provides information needed to make decisions about what a user is allowed to do. These decisions can be based on either the user's identity or role membership or both.

3 .Net framework in context

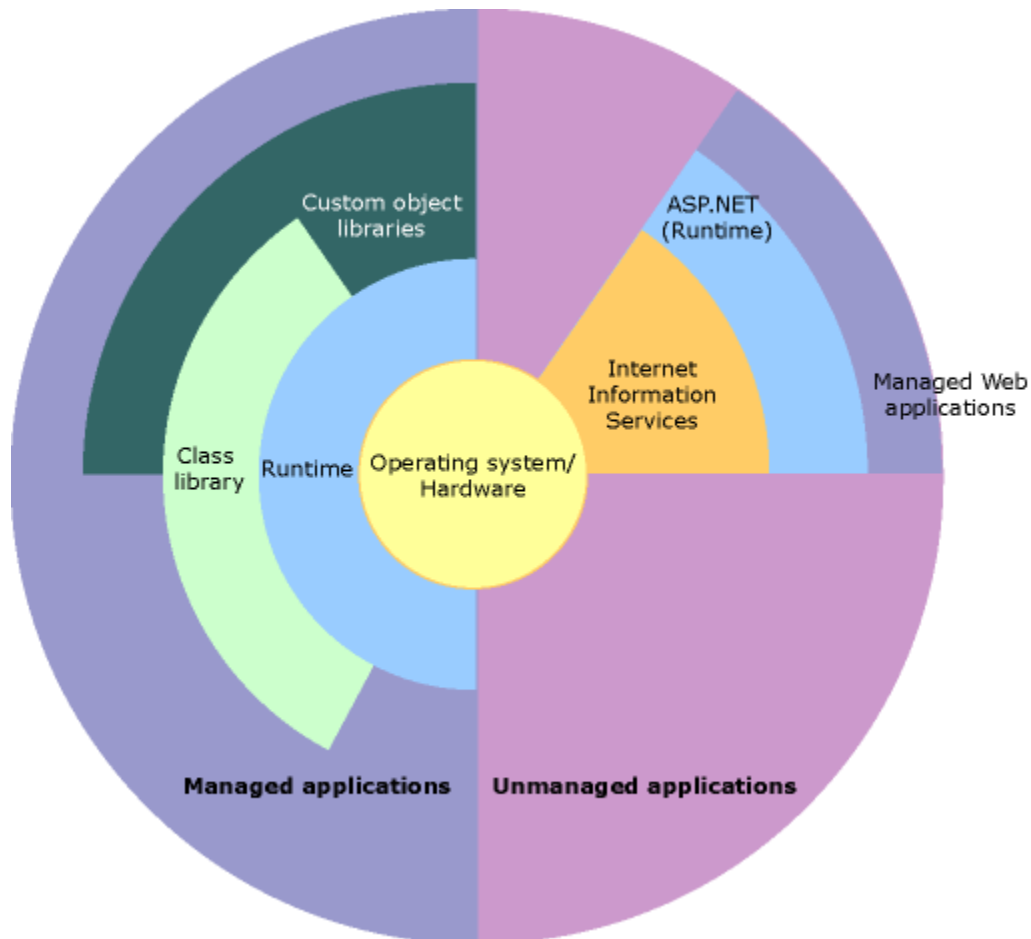


Figure 1: .Net framework in context

4. Automatic memory management

Automatic memory management is one of the services that the common language runtime provides during Managed Execution. The common language runtime's garbage collector manages the allocation and release of memory for an application. For developers, this means that they do not have to write code to perform memory management tasks when they

develop managed applications. Automatic memory management can eliminate common problems such as forgetting to free an object and causing a memory leak, or attempting to access memory for an object that has already been freed.

5. Allocating memory

When you initialize a new process, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the managed heap. The managed heap maintains a pointer to the address where the next object in the heap will be allocated. Initially, this pointer is set to the managed heap's base address. All reference types are allocated on the managed heap. When an application creates the first reference type, memory is allocated for the type at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner.

Allocating memory from the managed heap is faster than unmanaged memory allocation. Because the runtime allocates memory for an object by adding a value to a pointer, it is almost as fast as allocating memory from the stack. In addition, because new objects that are allocated consecutively are stored contiguously in the managed heap, an application can access the objects very quickly.

6. Releasing memory

The garbage collector's optimizing engine determines the best time to perform a collection based upon the allocations being made. When the garbage collector performs a collection, it releases the memory for objects that are no longer being used by the application. It determines which objects are no longer being used by examining the application's roots. Every application has a set of roots. Each root either refers to an object on the managed heap or is set to null. An application's roots include global and

static object pointers, local variables and reference object parameters on a thread's stack, and CPU registers. The garbage collector has access to the list of active roots that the just-in-time (JIT) compiler and the runtime maintain. Using this list, it examines an application's roots, and in the process creates a graph that contains all the objects that are reachable from the roots.

Objects that are not in the graph are unreachable from the application's roots. The garbage collector considers unreachable objects garbage and will release the memory allocated for them. During a collection, the garbage collector examines the managed heap, looking for the blocks of address space occupied by unreachable objects. As it discovers each unreachable object, it uses a memory-copying function to compact the reachable objects in memory, freeing up the blocks of address spaces allocated to unreachable objects. Once the memory for the reachable objects has been compacted, the garbage collector makes the necessary pointer corrections so that the application's roots point to the objects in their new locations. It also positions the managed heap's pointer after the last reachable object. Note that memory is compacted only if a collection discovers a significant number of unreachable objects. If all the objects in the managed heap survive a collection, then there is no need for memory compaction.

To improve performance, the runtime allocates memory for large objects in a separate heap. The garbage collector automatically releases the memory for large objects. However, to avoid moving large objects in memory, this memory is not compacted.

7. Overview of ADO.NET

ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed via OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data.

ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET data

providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET DataSet object in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remoted between tiers. The ADO.NET DataSet object can also be used independently of a .NET data provider to manage data local to the application or sourced from XML.

The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. When compiling code that uses the System.Data namespace, reference both System.Data.dll and System.Xml.dll.

ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native COM developers by ADO.

Chapter 3

POST OFFICE PROTOCOL 3

1. Introduction

A standard protocol for transferring mail messages on demand from a mail server. On certain types of smaller nodes in the Internet it is often impractical to maintain a message transport system (MTS). For example, a workstation may not have sufficient resources (cycles, disk space) in order to permit a SMTP server and associated local mail delivery system to be kept resident and continuously running. Similarly, it may be expensive (or impossible) to keep a personal computer interconnected to an IP-style network for long amounts of time (the node is lacking the resource known as "connectivity").

Despite this, it is often very useful to be able to manage mail on these smaller nodes, and they often support a user agent (UA) to aid the tasks of mail handling. To solve this problem, a node which can support an MTS entity offers a maildrop service to these less endowed nodes. The Post Office Protocol - Version 3 (POP3) is intended to permit a workstation to dynamically access a maildrop on a server host in a useful fashion. Usually, this means that the POP3 protocol is used to allow a workstation to retrieve mail that the server is holding for it.

POP3 is not intended to provide extensive manipulation operations of mail on the server; normally, mail is downloaded and then deleted. The term "**client host**" refers to a host making use of the POP3 service, while the term "**server host**" refers to a host which offers the POP3 service.

2. How does a POP3 mail server work?

POP3 mail servers store incoming email messages on the server until we download them. As messages are downloaded to our computer, they are deleted (optional) from the mail server and stored on the computer. ISPs commonly provide at least one (up to 10) POP3 email account as part of their Internet dial-up package, so chances are we already have one or more POP3 email addresses such as "ourname@ourisp.com".

3. Basic operation

Initially, the server host starts the POP3 service by listening on TCP port 110. When a client host wishes to make use of the service, it establishes a TCP connection with the server host. When the connection is established, the POP3 server sends a greeting. The client and POP3 server then exchange commands and responses (respectively) until the connection is closed or aborted.

A POP3 session progresses through a number of states during its lifetime. Once the TCP connection has been opened and the POP3 server has sent the greeting, the session enters the AUTHORIZATION state. In this state, the client must identify itself to the POP3 server. Once the client has successfully done this, the server acquires resources associated with the client's maildrop, and the session enters the TRANSACTION state. In this state, the client requests actions on the part of the POP3 server. When the client has issued the QUIT command, the session enters the UPDATE state. In this state, the POP3 server releases any resources acquired during the TRANSACTION state and says goodbye. The TCP connection is then closed.

A server must respond to an unrecognized, unimplemented, or syntactically invalid command by responding with a negative status indicator. A server must respond to a command issued when the session is in an incorrect state by responding with a negative status indicator. There is no general method for a client to distinguish between a server which does not implement an optional command and a server which is unwilling or unable to process the command.

A POP3 server may have an inactivity autologout timer. Such a timer must be of at least 10 minutes' duration. The receipt of any command from the client during that interval should suffice to reset the autologout timer. When the timer expires, the session does not enter the UPDATE state. The server should close the TCP connection without removing any messages or sending any response to the client.

4. States

4.1 *The AUTHORIZATION state*

Once the TCP connection has been opened by a POP3 client, the POP3 server issues a one line greeting. This can be any positive response. An example might be:

S: +OK POP3 server ready

The POP3 session is now in the AUTHORIZATION state. The client must now identify and authenticate itself to the POP3 server. Two possible mechanisms for doing this are the USER and PASS command combination and the APOP command. While there is no single authentication mechanism that is required of all POP3 servers, a POP3 server must of course support at least one authentication mechanism.

Once the POP3 server has determined through the use of any authentication command that the client should be given access to the appropriate maildrop, the POP3 server then acquires an exclusive access lock on the maildrop, as necessary to prevent messages from being modified or removed before the session enters the UPDATE state. If the lock is successfully acquired, the POP3 server responds with a positive status indicator. The POP3 session now enters the TRANSACTION state, with no messages marked as deleted. If the maildrop cannot be opened for some reason (for example, a lock cannot be acquired, the client is denied access to the appropriate maildrop, or the maildrop cannot be parsed), the POP3 server responds with a negative status indicator. (If a lock was acquired but the POP3 server intends to respond with a negative status indicator, the POP3 server must release the lock prior to rejecting the command.) After returning a negative status indicator, the server may close the connection. If the server does not close the connection, the client may either issue a new authentication command and start again, or the client may issue the QUIT command.

After the POP3 server has opened the maildrop, it assigns a message number to each message, and notes the size of each message in octets. The first message in the maildrop is assigned a message-number of "1", the second is assigned "2", and so on, so that the nth message in a maildrop is assigned a message-number of "n". In POP3

commands and responses, all message-numbers and message sizes are expressed in base-10 (i.e., decimal).

The summary for the QUIT command when used in the AUTHORIZATION state is given in appendix 'A'.

4.2 The TRANSACTION state

Once the client has successfully identified itself to the POP3 server and the POP3 server has locked and opened the appropriate maildrop, the POP3 session is now in the TRANSACTION state. The client may now issue any of the following POP3 commands repeatedly. After each command, the POP3 server issues a response. Eventually, the client issues the QUIT command and the POP3 session enters the UPDATE state.

The POP3 commands valid in the TRANSACTION state are given in appendix 'A'.

4.3 The UPDATE state

When the client issues the QUIT command from the TRANSACTION state, the POP3 session enters the UPDATE state. (Note that if the client issues the QUIT command from the AUTHORIZATION state, the POP3 session terminates but does NOT enter the UPDATE state.)

If a session terminates for some reason other than a client-issued QUIT command, the POP3 session does not enter the UPDATE state and must not remove any messages from the maildrop.

The summary for the QUIT command when used in the UPDATE state is given in appendix 'A'.

5. Optional POP3 commands

The POP3 commands mentioned above must be supported by all minimal implementations of POP3 servers. The optional POP3 commands on the other hand

permit a POP3 client greater freedom in message handling, while preserving a simple POP3 server implementation.

The summary of optional commands is given in appendix 'A'.

6. Scaling and operational considerations

Since some of the optional features were added to the POP3 protocol, experience has accumulated in using them in large-scale commercial post office operations where most of the users are unrelated to each other. In these situations and others, users and vendors of POP3 clients have discovered that the combination of using the UIDL command and not issuing the DELE command can provide a weak version of the "maildrop as semi-permanent repository" functionality normally associated with IMAP. Of course the other capabilities of IMAP, such as polling an existing connection for newly arrived messages and supporting multiple folders on the server, are not present in POP3.

When these facilities are used in this way by casual users, there has been a tendency for already-read messages to accumulate on the server without bound. This is clearly an undesirable behavior pattern from the standpoint of the server operator. This situation is aggravated by the fact that the limited capabilities of the POP3 do not permit efficient handling of maildrops which have hundreds or thousands of messages.

Consequently, it is recommended that operators of large-scale multi-user servers, especially ones in which the user's only access to the maildrop is via POP3, consider such options as:

- Imposing a per-user maildrop storage quota or the like. A disadvantage to this option is that accumulation of messages may result in the user's inability to receive new ones into the maildrop. Sites which choose this option should be sure to inform users of impending or current exhaustion of quota, perhaps by inserting an appropriate message into the user's maildrop.

- Enforce a site policy regarding mail retention on the server. Sites are free to establish local policy regarding the storage and retention of messages on the server, both read and unread. For example, a site might delete unread messages from the server after 60 days and delete read messages after 7 days. Such message deletions are outside the scope of the POP3 protocol and are not considered a protocol violation.

Server operators enforcing message deletion policies should take care to make all users aware of the policies in force. Clients must not assume that a site policy will automate message deletions, and should continue to explicitly delete messages using the DELE command when appropriate.

It should be noted that enforcing site message deletion policies may be confusing to the user community, since their POP3 client may contain configuration options to leave mail on the server which will not in fact be supported by the server.

One special case of a site policy is that messages may only be downloaded once from the server, and are deleted after this has been accomplished. This could be implemented in POP3 server software by the following mechanism: "following a POP3 login by a client which was ended by a QUIT, delete all messages downloaded during the session with the RETR command". It is important not to delete messages in the event of abnormal connection termination (i.e if no QUIT was received from the client) because the client may not have successfully received or stored the messages.

7. Algorithm for retrieving mail

The algorithm to retrieve mail from POP server is given in appendix 'A'.

8. Advantages of POP3 email

Most ISPs support POP3. After retrieval, the messages can be read offline. Messages can be composed offline and sent, the next time connection to the Internet is established. We can configure our mail client not to delete incoming mail from our POP3 account and keep a copy of our messages on the mail server. If we receive a lot of mail,

this is not a good idea since messages piling up at the mail server can easily exceed disk storage quotas. POP3 accounts can be accessed from any PC and Internet connection at any time. There are literally hundreds of other advantages using POP3 email accounts mainly due to the flexibility of organizing, managing, and filtering our messages on our computer.

Chapter 4

MESSAGING APPLICATION PROGRAMMING INTERFACE

1. Introduction

Before getting into the details of how the Messaging Application Programming Interface (MAPI) works and how to write MAPI applications, we'll take a moment to review the general architecture of Microsoft's messaging API and how this set of message services fits into the overall Windows operating system. MAPI is more than a handful of e-mail APIs. It is a defined set of message services available to all programs that run in the Windows operating environment.

We'll also discuss the various kinds of applications and message types commonly used under MAPI services. In this chapter, we will see the three general types of MAPI messages: text messages, formatted documents and binary files, and control messages. Each of these message types has a distinct set of properties and uses within the MAPI framework. This chapter describes each of the message types and shows how we can use them within the MAPI architecture.

There are also three common types of MAPI applications: e-mail clients, message-aware applications, and message-enabled applications. Each of these application types is defined and illustrated in this chapter. We will also see the relative strengths of each type of MAPI application.

2. MAPI services and windows

The MAPI service set is more than a set of API commands and functions that we can use to send e-mail from point to point. The MAPI interface is actually a carefully defined set of messaging services available to all Windows programs. This pre-defined set has three key attributes:

- Flexibility
- Consistency
- Portability

Because the MAPI service set contains these three characteristics, it has become the *de facto* messaging interface standard for Windows applications. Access to MAPI services is the same for all versions of the Windows operating system. But even though

Windows programs use the same methods for accessing MAPI services, MAPI services can vary from system to system. Also, MAPI architecture allows software designers to create their own service providers (SPs) to support the MAPI service set. These services are also available within all existing flavors of the Windows operating system. Even more important, the methods to access these message services is the same, regardless of the version of Windows we are working with. This means that programs using MAPI services that were written under Windows 98 will still be able to access those same MAPI services under Windows 2000.

2.1 Flexibility

Probably the most important aspect of the MAPI service architecture is its flexibility. Microsoft has implemented MAPI within the Windows Open Systems Architecture (**WOSA**). This architecture is designed to allow customization at both the client (user) side and the server (provider) side. In other words, we can use MAPI not only to create our own end-user software to read, write, create, and send messages, but also to construct custom server-side software to store and transport those same messages. As part of the WOSA model, MAPI services are implemented in a tiered format.

2.1.1 The tiered implementation of MAPI services.

The first layer is the *client* layer. This is what the end-user most often sees. At this level a set of well-defined services are available. These services are accessed when the client layer makes a service request to the second layer-the *MAPI DLL*. The MAPI DLL takes the service request from the client and forwards it on to the third layer in the tier-the *service provider*. The service provider is responsible for fulfilling the client request and sending the results of that request back to the DLL where it is then forwarded to the client that made the initial request. Throughout the process, the DLL layer acts as a broker between the client side and the server side.

The primary advantage of this layered implementation is the ease with which users can interchange client and server components. Since the only constant required in the mix is the DLL layer, any client can be matched with any server component to provide a working message service.

2.2 Consistency

The MAPI service set contains a set of services that encompasses all the basic messaging tasks:

- Message service logon and logoff.
- Reading, creating, and deleting text messages.
- Adding and deleting message binary file attachments.
- Addressing and transporting the completed messages.

The exact behavior and properties of each of these services are defined as part of MAPI. This way, any program that uses MAPI services can be assured that there are no special API variations to deal with when moving from one vendor's MAPI product to another. This means the programs we write today using our current implementation of MAPI services will function under other implementations of the MAPI service set.

2.3 Portability

This leads to the third strength of Microsoft's MAPI service set—portability. Microsoft supports MAPI services on all versions of its Windows operating systems. If we write a program for the Windows 95 version of MAPI services, that same program can still access the MAPI services under any other version of Windows that supports our executable program. This is a key issue when we consider how many versions of Windows are currently in use and how quickly new versions of the operating system are developed and deployed.

Not only will we be able to move our MAPI-related programs to various Windows platforms, we can also allow programs to access MAPI services from more than one platform at once. In other words, users of more than one version of Windows can all be accessing MAPI services from a central location at the same time. Microsoft has announced plans to move several of its service sets (MAPI included) beyond the Windows operating environment, too. As this happens, Microsoft has pledged that the same set of functions and routines used under the Windows environment will be available in other operating systems.

3. Messages

The primary role of the MAPI service is to transport and store messages. This section identifies three common message types supported by MAPI services:

- Text messages
- Formatted documents or binary files
- Control messages

The most basic message form is the text message, commonly thought of as e-mail. Most electronic message systems also support the second type of message-formatted documents or binary files. These are usually included as attachments to a text message.

The third message type is a *control message*. Control messages are usually used by the operating system to pass vital information such as system faults, potential failure conditions, or some other type of status information. Control messages can also be passed between programs in order to implement a level of distributed processing in a computer network.

3.1 Text messages

The text message is the most common MAPI message. In fact, all MAPI messages have a default text message component. A text message contains the letters and words composed by users to communicate with other message system users.

All MAPI service providers must supply a simple text message editor as part of their MAPI implementation. All MAPI message providers support plain ASCII text characters as a message body. Many also support rich-text messages that contain formatting characters such as font and color.

3.2 Formatted documents and binary files

The second MAPI message type is the formatted document or binary file, which is usually a file containing non-printable characters such as a spreadsheet, a word-processing file, graphics, or even an executable program. Binary files are supported by MAPI services as attachments to text messages. The MAPI service set allows multiple

attachments to a single text message. This means we can send several binary files to the same e-mail address using a single message body.

All MAPI service providers support the use of binary attachments to a message body. However, the transport of binary attachments across multiple message servers may not be supported. For example, if we compose a message that contains attached binary files, address it to an associate at a distant location, and attempt to send the message using a service provider that supports only Simple Mail Transfer Protocol (SMTP) format, our attached binary files will not be successfully transported to the recipient.

3.3 Control messages

The third type of MAPI message is the control message. Control messages are usually used by the operating system to deliver system status information, such as a component failure or other system-related problem. These messages are usually addressed directly to the system administrator.

Control messages can also be used to pass data or other control information between programs. Control messages of this type can contain requests for information that is to be collected by one program and returned to another for further processing. Or the control message can contain actual data to be manipulated by another program. Since MAPI services can stretch across the LAN or across the globe, control messages can be passed to systems halfway around the globe as easily as to systems across the room.

It is possible to designate one or more workstations on a network as *batch job* computers. These machines wait for control messages that direct them to perform time-consuming tasks, such as extended database searches or generating long reports, thus freeing up users' workstations for more immediate business. Once the task is complete, the batch job machine can send a completion notice via e-mail to the user who sent the original request.

4. MAPI applications

Just as there are three types of MAPI messages, there are three general types of MAPI applications:

- Electronic mail clients
- Message-aware applications
- Message-enabled applications

4.1 Electronic mail clients

Electronic mail (e-mail) clients are the most common form of MAPI application. An e-mail client allows end-users direct access to the MAPI services supported by the back-end service provider.

Typical services provided by a MAPI e-mail client include

- Message service logon and logoff.
- Reading, creating, and deleting text messages.
- Adding and deleting binary file message attachments.
- Addressing and transporting completed messages.

Electronic mail clients can also provide additional services to make it easy to manipulate, store, and retrieve text messages and binary file attachments. Electronic mail clients may also have additional features for addressing and transporting messages, including the use of defined mailing lists and the capability to address messages as cc (courtesy copies) or Bcc (blind courtesy copies).

4.2 Message aware applications

Message-aware applications are non-MAPI programs that allow users access to MAPI services. Typically, this access is implemented through the addition of a send option in a menu or button bar.

Message-aware applications usually treat e-mail services just like any other storage or output location, such as disk drives, printers, or modems. In these cases, the ability to send the standard output as an electronic mail message is an added feature for the application. As MAPI services become a standard part of the Windows operating system, message-aware applications will become the norm instead of the exception.

4.3 Message enabled applications

The last category of MAPI applications is message-enabled applications. Message-enabled applications are programs that offer message services as a fundamental feature. While message-aware applications provide message services as an additional feature and can operate well without them. Message-enabled applications are specifically designed to use message services and most will not run properly unless message services are available on the workstation.

Here are some examples of message-enabled applications:-

- ***Computerized service dispatch.*** Customer calls are handled by representatives at PC workstations where they fill out data entry forms outlining the repair needs and the location of the service call. When the data entry is complete, the program analyzes the information and, based on the parts needed and the service location, routes an instant electronic message containing the service request and a list of needed parts to the repair office nearest to the customer.
- ***Online software registration.*** When a user installs a new software package, part of the installation process includes an online registration form that already contains the unique software registration code along with a data entry form for the user to complete. Once the form is completed, the results are placed in the user's e-mail outbox to be sent directly to the software company to confirm the user's software registration.
- ***End-user support services.*** When network end-users have a question about a software package or need to report a problem with their workstation or the network, they call up a data entry form prompting them to state the nature of the problem. This program will also automatically load the user's system control files and add them as attachments to the incident report. Once the form is complete, it is sent (along with the attachments) to the appropriate network administrator for prompt action.

It is important to note that, in some cases, users of message-enabled applications may not even be aware that they are using the e-mail system as part of their application.

MAPI services define properties and methods for logging users in and out of the message server without using on-screen prompts. MAPI also provides options for addressing and sending messages without the use of on-screen prompts or user confirmation. By using these features of MAPI services, we can design a program that starts a message session, reads mail, composes replies, addresses the new mail, and sends it to the addressee without ever asking the user for input.

5. Other types of message applications

There are two more types of message-enabled applications that deserve comment here. These two application types are:

- Electronic forms applications
- Message-driven applications

Electronic forms applications display a data entry screen that contains one or more data fields for the user to complete. These data fields act as full-fledged windows controls and can support all the events normally supported by Windows data entry forms. Once the form is completed, the data, along with additional control information, is sent to the addressee through MAPI services. When the addressee opens the new mail, the same formatted data entry form appears with the fields filled in.

The message-driven application looks for data contained in a message and acts based on the data it finds. Message-driven applications can use any aspect of the message as control information for taking action. Message-driven applications can inspect the message body or subject line for important words or phrases, check the sender's name or the date and time the message was sent, or even scan attachments for important data. These applications can then use the data to forward messages to another person automatically, to set alerts to notify the user of important messages, or to start other programs or processes at the workstation.

Below are some examples of message driven applications:-

- ***Message filtering agent.*** Users can enter a list of important keywords into a list box. This list is used to scan all incoming text messages automatically. If the message contains one or more of the keywords, the user is notified immediately that an important message has arrived. Users

could also set values to scan for the message sender's name. For example, if the message came from the user's boss, an alert could sound to warn the user that an urgent message has arrived. The same technique can be used to automatically forward specific messages when the user is away on a trip.

- ***File transfer database update application.*** This program could be used by outlying sales offices to update a central database automatically. Each day the remote offices would enter sales figures in a database, then attach the binary database file to an e-mail message, and send the message to the corporate headquarters. There, a special workstation (logged in as the addressee for all sales database updates) would receive the message and automatically run a program that takes the binary database file and merges it into the central database. This same program could then provide summary data back to the remote offices to keep them up to date on their progress.
- ***Electronic database search tool.*** Many companies have large libraries of information on clients, products, company regulations, policies and procedures, and so on. Often users would like to run a search of the information but don't have time to physically visit the library and pour through thousands of pages in search of related items. If the information is kept in online databases, users at any location around the world could formulate a set of search criteria to apply against the databases and then submit these queries, via MAPI messages, to one or more workstations dedicated to performing searches. After the search is completed, the resulting data set could be returned to the user who requested the data.

Filtering agents, remote update routines, and long-distance search tools are all examples of how MAPI services can be used to extend the reach of the local workstation to resources at far-away locations. The Windows MAPI services provide excellent tools for building programs that enable users to collect and/or disseminate data over long distances or to multiple locations.

6. MAPI architecture

Now we will see the basic conceptual components of MAPI, the MAPI Client and the MAPI Server. These two components work together to create, transport, and store messages within the MAPI system.

MAPI clients deal with three main objects:-

- Messages and attachments
- Storage folders
- Addresses

Each of these objects is represented slightly differently in the various versions of MAPI implementations. For example, the MAPI OCX tools that ship with Visual Basic allow only limited access to folders and address information. The Messaging OLE layer (provided through the MAPI 1.0 SDK) provides increased access to folders and addresses, but only the full MAPI 1.0 functions allow programmers to add, edit, and delete folders and addresses at the client level.

MAPI servers also deal with three main objects:-

- Message transport
- Message stores
- Addresses books

Where the client is concerned with creating and manipulating messages, the server component is concerned with the transporting of those same messages. Where the client side is accessing storage folders, the server side is dealing with message storage, and both client and server must deal with message addresses. However, the MAPI server has the responsibility of managing the transport, storage, and addressing of messages from any number of client applications.

In addition to maintaining the message base for all local clients, MAPI servers also have the task of moving messages to and from remote servers and clients. Finally we will briefly talk about a special MAPI component that handles this task-the MAPI Spooler.

7. The MAPI client

The MAPI Client is the application that runs on the user's workstation. This is the application that requests services from the MAPI Server. As mentioned earlier, client applications can be generic e-mail tools such as Microsoft's Microsoft Mail or Exchange Mail Client for Windows 95. Client applications can also be message-aware applications like the Microsoft Office Suite of applications. Each of these applications provides access to the message server via a send menu option or command button. Lastly, message-enabled applications, ones that use MAPI services as a primary part of their functionality, can be built to meet a specific need. These programs usually hide the MAPI services behind data entry screens that request message-related information and then format and send messages using the available message server.

All MAPI clients must, at some level, deal with three basic objects:

- Messages and attachments
- Storage folders
- Addresses

Depending on the type of client application, one or more of these MAPI objects may be hidden from the client interface. However, even though they may not be visible to the application user, all three objects are present as part of the MAPI architecture.

7.1 Messages and attachments

The MAPI system exists in order to move messages from one location to another. Therefore, the heart of the system is the MAPI message object. All message objects must have two components—a message header and a message body. The message header contains information used by MAPI to route and track the movement of the message object. The message body contains the actual text message portion of the message object. Even though every message object must have a message body, the body can be left blank. In addition to the two required components, message objects can also have one or more attachments. Attachments can be any valid operating system file such as an ASCII text file, a binary image, or an executable program.

7.1.1 *The message header*

The Message Header contains all the information needed to deliver the associated message body and attachments. Data stored in the message header varies, depending on the messaging service provider. While the exact items and their names and values differ between messaging systems (CMC, MAPI, OLE Messaging), there is a basic core set that appears in all message headers. Examples of basic data items that can be found in a message header are listed in Table 1.

<i>Note</i>
<i>The item names given in Table 1 do not necessarily correspond to a valid property or variable name in programming code. The actual property names for each item can vary from one code set to another. Also, the order in which these properties appear differs greatly for each MAPI implementation.</i>

<i>Property Name</i>	<i>Type</i>	<i>Description</i>
Recipients	Recipients object	E-mail address of the person who will receive the message. This could be a single name, a list of names, or a group name.
Sender	AddressEntry object	E-mail address of the person who sent the message.
Subject	String	A short text line describing the message.
TimeReceived	Variant (Date/Time)	The date and time the message was received.
TimeSent	Variant (Date/Time)	The date and time the message was sent.

Table 1: Basic data items found in a message header

Along with this basic set, additional data items may appear or be available to the programmer. These additional items add functionality to the MAPI interface, but because they are not part of the core set, they may not be available while writing programs. Table 2 contains a list of additional header data items.

<i>Property Name</i>	<i>Type</i>	<i>Description</i>
DeliveryReceipt	Boolean	Flag that indicates the sender asked for a return receipt message upon either delivery of the message to the recipient or the reading of the message by the recipient.
Importance	Long	A value that indicates the relative importance of the message. Currently, Microsoft Mail clients recognize three priorities: High, Medium, and Low.
Submitted	Boolean	Flag that indicates the item has been sent to the recipient.
Sent	Boolean	Read/write.
Signed	Boolean	Read/write.
Type	String	Value that identifies this message as one of a class of messages. Currently, Microsoft Mail systems recognize the IPM (Interpersonal Message) type for sending messages read by persons. Microsoft has defined the Ipc (Interprocess Communication) type for use between program processes. Other types can be defined and used by other programs.
Unread	Boolean	Flag that indicates whether the message has been received and/or read by the recipient.

Table 2: Optional items that may be found in the message header

7.1.2 The Message body

The message body contains the text data sent to the recipient from the sender. For most systems, this is a pure ASCII text message. However, some service providers can handle rich-text format message bodies, which allows for additional information such as font, color, and format codes to be included in the message body.

The availability and support of rich-text message bodies vary between service providers. Some service providers allow us to create rich-text message bodies but translate the information into simple ASCII upon delivery. For example, Microsoft Exchange allows users to build rich-text message bodies but translates that message into simple ASCII text when using the SMTP service provider. All messages received by the Microsoft Mail system are converted into simple text as well. This behavior ensures that the message will be delivered but may result in surprise or even unreadable material at the recipient's end

Other transport providers may transmit the rich-text and allow the receiving message provider to handle the translation to simple ASCII, if needed. This option allows for the most flexibility but can result in undeliverable messages. For example, Microsoft Exchange users have the option of sending rich-text messages through the CompuServe message transport. The CompuServe transport supports rich-text messages. Rich-text messages sent from one Windows Messaging Client to another by way of the CompuServe transport retain their original layout and look. However, recipients using something other than Microsoft Exchange may see something different.

7.1.3 Message attachments

Message attachments are supported by all forms of Microsoft MAPI. A MAPI attachment can be any data file, of any type (text, binary programs, graphics, and so on). These attachments are sent to the recipient along with the message header and body. Upon receipt of the message, the recipient can, depending on the features of the message client software, view, manipulate, and store the attachments on the local workstation. The MAPI system keeps track of attachments with a set of properties. Table 3 lists an

example set of attachment properties. The actual properties and their names can differ between program code sets.

<i>Property Name</i>	<i>Type</i>	<i>Description</i>
Index	Long	Each message object can contain more than one attachment. Attachments are numbered starting with 0.
Name	String	The name to display in a list box or in the client message area (for example, "June Sales Report").
Position	Long	A value that indicates where in the message body the attachment is to be displayed. Microsoft Mail and Windows Messaging clients display an icon representing the attachment within the message body. Other clients may ignore this information, show an icon, or show ASCII text that represents the attachment.
Source	String	The exact filename used by the operating system to locate and identify the attachment (for example, "\\Server1\Data\Accounting\JuneSales.xls").
Type	Long	A value that indicates the type of attachment. Microsoft defines three attachment types: <i>Data-A direct file attachment</i> <i>Embedded OLE-An embedded OLE object</i> <i>Static OLE-A static OLE object</i>

Table 3: Example MAPI attachment properties

Attachments can be handled differently by the message transport provider. Microsoft Mail and Windows Messaging clients display attachments within the message body and transport the attachments as part of the message body, too. Microsoft Mail and

Microsoft Exchange recipients see the attachment within the message body and can use the mouse to select, view, and save the attachment when desired. Other transports may handle the attachment differently.

7.2 *Storage folders*

MAPI messages can be saved in storage folders. The MAPI model defines following storage folders:-

- ***Inbox.*** This is the place where all incoming messages first appear.
- ***Outbox.*** This is the place where all outgoing messages are placed before they are sent to their destination.
- ***Sent.*** This is the place where all outgoing messages are placed after they are sent to their destination. This is, in effect, a set of message copies that can be referenced after the original has been sent.
- ***Deleted.*** This is the place where all messages are placed once they have been marked for deletion.
- ***User-defined folders.*** This can be one or more folders defined by the user. Each folder can hold messages that have been received and copies of messages that have been sent.

Not all implementations of MAPI support all of the folders listed above. For example, the Simple MAPI interface allows access to the `Inbox` (by way of the `.Fetch` method of the `MAPISESSION` Control) and the `Outbox` (by way of the `.Send` method of the `MAPISESSION` Control). Simple MAPI allows no other folder access. Programmers cannot inspect the contents of the `Sent` folder or move messages from the `Inbox` to other user-defined storage folders.

The MAPI model defines a handful of properties for storage folders. Table 4 lists some of the more commonly used properties of the storage folder.

<i>Property Name</i>	<i>Type</i>	<i>Description</i>
FolderID	String	This is a string value that uniquely identifies this folder.
Folders	Folders collection object	This is the set of folder objects contained by the current folder. Any folder can contain one or more sublevel folders.
Messages	Messages collection object	This is the set of messages stored in this folder.
Name	String	A unique user-defined string that identifies the storage folder.
Parent	Object	This contains the name of the parent folder or InfoStore to which the current folder belongs.

Table 4: Example MAPI storage folder properties

7.3 *Addresses*

Addresses are the last class of objects dealt with at the client level. Every electronic message has at least two address objects: the sender object and the recipient object. MAPI allows us to add several recipient address objects to the same message. Each address object has several properties. Table 5 shows a set of sample properties for the MAPI `Address` object.

<i>Property Name</i>	<i>Type</i>	<i>Description</i>
Address	String	This is the unique electronic address for this address object. The combination of the <code>Type</code> property (see below) and the <code>Address</code> property creates the complete MAPI address. Sample address properties are Asif@isp.net-Internet address /MailNet1/PostOfc9/MCA-MS Mail address
DisplayType	Long	The MAPI service allows programmers to define addresses by type. This means you can sort or filter messages using the <code>DisplayType</code> property. Sample address types are mapiUser-Local user mapiDistList-Distribution list mapiForum-Public folder mapiRemoteUser-Remote user
Name	String	This is the name used in the Address book. Usually, this is an easy-to-remember name such as "Fred Smith" or "Mary in Home Office."
Type	String	This value contains the name of the message transport type. This allows MAPI to support the use of external message transport services. Sample address types are MS:-Microsoft Mail transport SMTP:-Simple Mail Transport Protocol MSN:-Microsoft Network transport

Table 5: MAPI Address object properties

8. The MAPI server

The MAPI Server handles all the message traffic generated by MAPI clients. The MAPI Server usually runs on a standalone workstation connected to the network, but this

is not a requirement. There are versions of user-level MAPI servers that can be used to handle message services.

Microsoft supports two standalone MAPI servers:

- **Microsoft Mail Server** (for both PCs and Apple workstations)
- **Microsoft Exchange Server** (for NT Server workstations)

The Microsoft Mail Server runs standalone on both Intel PCs or Apple workstations. It provides direct MAPI services for all connected MAPI users and also provides gateway MAPI services for remote users. The Microsoft Mail Server has, until recently, been Microsoft's primary electronic mail server. Even though Microsoft is stressing the early adoption of the new Microsoft Exchange Server for NT, the Microsoft Mail Server will continue to be the primary mail server for thousands of users. All MAPI Clients can share information with connected Microsoft Mail Servers regardless of the client platform.

The Microsoft Exchange Server runs as a service on an NT Server workstation. It provides MAPI services to all MAPI users. Unlike the Microsoft Mail Server, which distinguishes between local and remote users, the Microsoft Exchange Server treats all MAPI users as remote users. This simplifies several aspects of MAPI administration. Unlike the Microsoft Mail Server, which only supports Microsoft Mail format messages, the Microsoft Exchange Server supports multiple message formats and services, including Microsoft Mail. This also means that the administration of gateways and remote transports is quite different for Microsoft Exchange.

Regardless of the actual server application used, the same basic processes must occur for all MAPI server systems. The three main tasks of all MAPI servers are:-

- ***Message transport.*** Moving the message from location to location.
- ***Message storage.*** Providing a filing system for the storage and retrieval of received messages.
- ***Address book services.*** Providing centralized addressing and verification services that can be used by all MAPI clients.

8.1 *Message transport*

Message Transport is the process of moving messages from one place to another. Under the MAPI model, message transport is a distinct, and often separate process. MAPI 1.0 allows for the use of *external message transports*. In other words, programmers can write software that knows how to handle a particular type or types of message formats and register this transport mechanism as part of the MAPI system. This allows third-party vendors to create format-specific transports that can be seamlessly integrated into the MAPI system.

It is the message transport that knows just how to format and, if necessary, pre-process messages for a particular messaging format. The message transport knows exactly what information must be supplied as part of the message header and how it needs to be arranged. The message transport also knows what types of message bodies are supported. For example, SMTP format allows only text message bodies. However, the Microsoft Network message format allows rich-text message bodies. It is the job of the message transport to keep track of these differences, modify the message where appropriate, or reject the message if modification or pre-processing is not possible.

One of the key features of the MAPI model is the provision for multiple message transports within the MAPI system. Once message transports are installed (or registered) with a MAPI client application, they are called into action whenever the pre-defined message type is received by the MAPI client software. Since MAPI is designed to accept the registration of multiple transports, the MAPI Client is potentially capable of handling an unlimited number of vendor-specific message formats.

Note

Message types are stored as part of the address. These types were discussed earlier in this chapter in the "Addresses" section.

Under the MAPI system, message transports provide another vital service. It is the responsibility of the message transport to enforce any security features required by the

message format. For example, the MSN mail transport is responsible for prompting the user for a username and password before attempting to link with the MSN mail system.

It is important to note that the message transport is not responsible for storing the messages that have been received. The transport is only in charge of moving messages from one location to another.

8.2 *Message stores*

Message stores are responsible for providing the filing system for the messages received via the message transport. The MAPI model dictates that the message store must be in a hierarchical format that allows multilevel storage. In other words, the system must allow users to create folders to hold messages, and these folders must also be able to hold other folders that hold messages. Under the MAPI model, there is no limit to the number of folder levels that can be defined for a message store.

Under the MAPI model, storage folders can have properties that control how they are used and how they behave. For example, storage folders can be public or private. Folders can have properties that make the contained messages read-only to prevent modification. The options available depend on the implementation of the message store. In other words, the programmer who designs the message store can establish the scope of storage options and the MAPI Client will comply with those rules.

As in the case with message transports, MAPI clients can access more than one message store. The Windows Messaging Client that ships with Microsoft Exchange Server also allows us to create folder column, grouping, sort, and filter rules for personal and public folders. By doing this, we can create storage views that reflect the course of an ongoing discussion and allow for easy search and retrieval of data kept in the message store.

8.3 *Address books*

The last of the main MAPI server objects is the *address book*. The MAPI address book contains all the directory information about a particular addressee. The book can contain data for individual users or groups of users (a distribution list). The minimum

data stored in the address book is the user's display name, the transport type, and the user's e-mail address. Additional information such as mailing address, telephone number, and other data may be available depending on the design of the address book.

Address books, like the other objects described earlier, work independently under the MAPI model. Also, the MAPI client can access more than one address book at a time. This means that several address books of various formats can all be viewed (and used) at the same time when composing messages.

Alongwith storing addresses, the address book interface also acts to resolve display names used in the MAPI interface with the actual e-mail addresses and transport types for those display names. To do this, MAPI offers a ResolveName service that performs lookups upon request. The ResolveName service is able to look at all address books (regardless of their storage format) in order to locate the proper e-mail address.

Users are also able to designate one of the address books as the default or *personal* address book. This is the first address book in which new addresses are stored and the first address book that is checked when resolving a display name. The Windows Messaging Client and the Microsoft Mail client both ship with default personal address books. The Windows Messaging Client allows users to add new address books and designate their own personal address book container.

9. The MAPI spooler

The MAPI Spooler is a special process that interacts with both message stores and message transports. It is the spooler's job to route messages from the client to the proper transport and from the transport to the client. The spooler is the direct link between the client and the transport. All messages go through the MAPI Spooler.

Note

Actually there are some cases in which a message moves directly from the message store to the message transport. This occurs when service providers offer both message store and message transport. E-mail service providers that offer these features are known as tightly

coupled service providers.

As each message is moved from the message store (the "outbox") to the transport, the MAPI Spooler checks the address type to see which transport should be used. Once this is determined, the spooler notifies the transport and attempts to pass the message from the message store to the message transport. If the transport is not currently available, the MAPI Spooler holds onto the message until the transport is free to accept messages. This allows transport providers to act as remote connections without any additional programming or addressing on the client side.

Note

In fact, the implementation used in the Microsoft Exchange version of MAPI treats all connections as if they were remote—even when the message is moved from one user's Microsoft Exchange outbox to another Microsoft Exchange user's inbox on the same network.

In the case of messages that move along a constantly connected transport (that is, between two addresses on the same Microsoft Exchange Server), the spooler notifies the transport (Microsoft Exchange) and the transport accepts the message almost immediately. Often the user is not aware of any delay in the handling of the message.

In the case of messages that move from the Windows Messaging Client by way of an SMTP transport through a dial-up connection to the Internet, the MAPI Spooler holds onto the message until the user connects to the Internet account. Once the connection is made, the MAPI Spooler sends all local messages on to the Internet mail server, retrieves any waiting mail from the mail server, and passes these new messages to the appropriate message store.

The MAPI Spooler is also able to move a single message to several recipients when some of those recipients are not using the same message transport. For example, users can build distribution lists that contain names of users on the local Microsoft Exchange Server, users who have addresses on a local Microsoft Mail Server, and users

who can only be contacted through a fax address. When the message is sent, it moves from the message store to the spooler, which then sorts out all the transports needed and passes the messages on to the correct transports at the first available moment.

The MAPI Spooler is also responsible for marking messages as read or unread, notifying the sender when a message has been successfully passed to the transport, and, when requested, notifying the sender when the recipient has received (or read) the message. The MAPI Spooler also reports when messages cannot be sent due to unavailable transports or other problems.

Chapter 5

TELEPHONY APPLICATION PROGRAMMING INTERFACE

1. Introduction

The *Telephony Application Programming Interface* (TAPI) is one of the most significant API sets to be released by Microsoft. The telephony API is a single set of function calls that allows programmers to manage and manipulate any type of communications link between the PC and the telephone line(s). While telephony models for the PC have been around for several years, the telephony API establishes a uniform set of calls that can be applied to any type of hardware that supplies a TAPI-compliant service provider interface (SPI).

2. The telephony API model

The telephony API model is one designed to provide an abstracted layer for access to telephone services from all Windows platforms. In other words, the telephony API is a single set of functions that can be used to access all aspects of telephony services within the Windows operating system.

This is a huge undertaking. The aim of TAPI is to allow programmers to write applications that work regardless of the physical telephone medium available to the PC. Applications written using TAPI to gain direct access to telephone-line services work the same on analog or digital phone lines. Applications that use TAPI can generate a full set of dialing tones and flash-hook functions (like that of the simple analog handset found in most homes), and can also communicate with sophisticated multi-line digital desktop terminals used in high-tech offices.

The TAPI design model is divided into two areas, each with its own set of API calls. Each API set focuses on what TAPI refers to as a *device*. The two TAPI devices are:-

- *Line devices* to model the physical telephony lines used to send and receive voice and data between locations.
- *Phone devices* to model the desktop handset used to place and receive calls.

2.1 Lines

The line device is used to model the physical telephone line. It is important to understand that, in TAPI, the line device is not really a physical line; it's just a model or object representing a physical line. In TAPI applications, a program could keep track of several line devices, each of which is connected to a physical line. That same TAPI application could also keep track of multiple line devices that number more than the total physical lines available to the PC.

For example, a single TAPI application could be designed to provide voice, fax, and data links for a user. The TAPI application would identify three line devices. One for voice calls, one for fax transmission, and one for sending and receiving data via an attached modem. If the PC has only one physical phone line attached, the TAPI application would share the one line between the three defined line devices. This is called *dynamic line mapping* (see Figure 1).

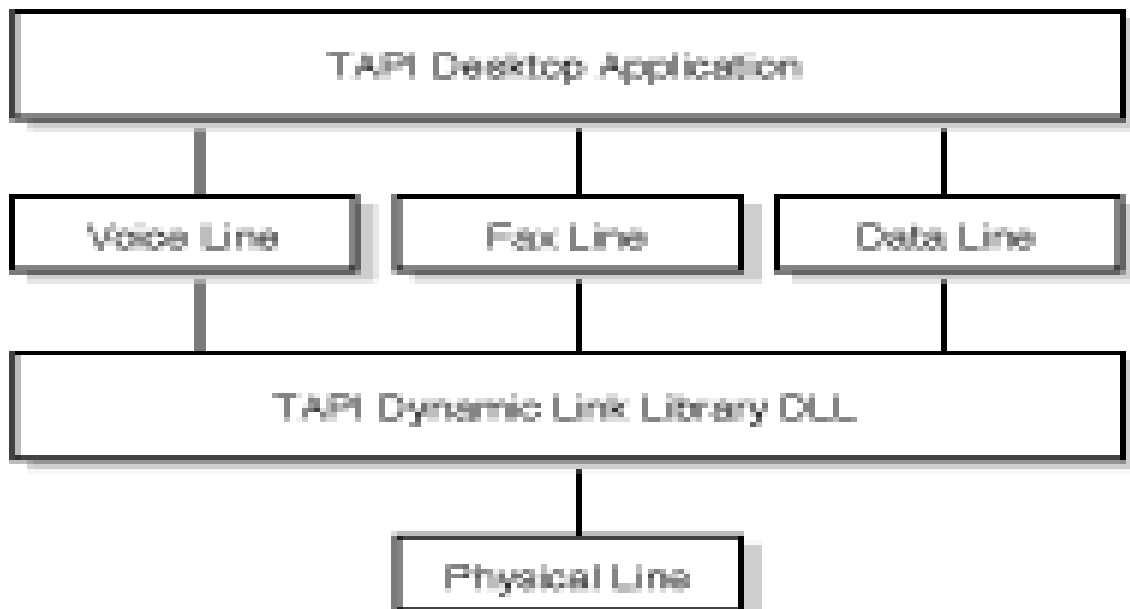


Figure 1: Dynamic line mapping

2.2 *Phones*

The second type of device modeled by TAPI is the phone device. This model allows TAPI programmers to easily create "virtual phones" within the PC workspace. For example, a standard PC with a sound card, speakers, and microphone can emulate all the functions of a desktop phone. These virtual phones, like their line device counterparts, need not exist in a one-to-one relationship to physical phones. A single PC could model several phone devices, each with their own unique characteristics. When an actual call must be made, the user could select one of the phone devices, enter the desired number and then the TAPI application would attach the phone device to an available line device. Note that the phone devices link to line devices (which eventually link to physical telephone lines). One of the primary uses of multiple phone devices would be the modeling of an office switchboard.

3. TAPI and the WOSA model

TAPI is able to accomplish its task by dividing the job into two distinct layers: the client API and the SPI. Each interface is a set of functions designed to complete generic telephony tasks such as opening a line, checking for a dial tone, dialing a number, checking for a ring or a busy signal, and so on. The client API sends requests from the application to the SPI for each task. It is the job of the SPI to complete the task and pass the results back to the calling program through the client API.

4. *Typical configurations*

The TAPI model is designed to function in several different physical configurations, which each have advantages and drawbacks. There are four general physical configurations:-

- *Phone-based.* This configuration is best for voice-oriented call processing where the standard handset (or some variation) is used most frequently.
- *PC-based.* This configuration is best for data-oriented call processing where the pc is used most frequently for either voice or data processing.

- **Shared or unified line.** This is a compromise between phone-based and PC-based systems. It allows all devices to operate as equals along the service line.
- **Multi-line.** There are several variations of the multi-line configuration. The primary difference between this configuration and the others is that the PC acts as either a voice-server or a call switching center that connects the outside phone lines to one or more PCs and telephone handsets. The primary advantage of multi-line configurations is that you do not need a direct one-to-one relationship between phone lines and end devices (phones or PCs).

4.1 Phone-based configurations

In phone-based TAPI configurations, the standard telephone handset is connected to the telephone switch and the PC is connected to the telephone (see Figure 2).

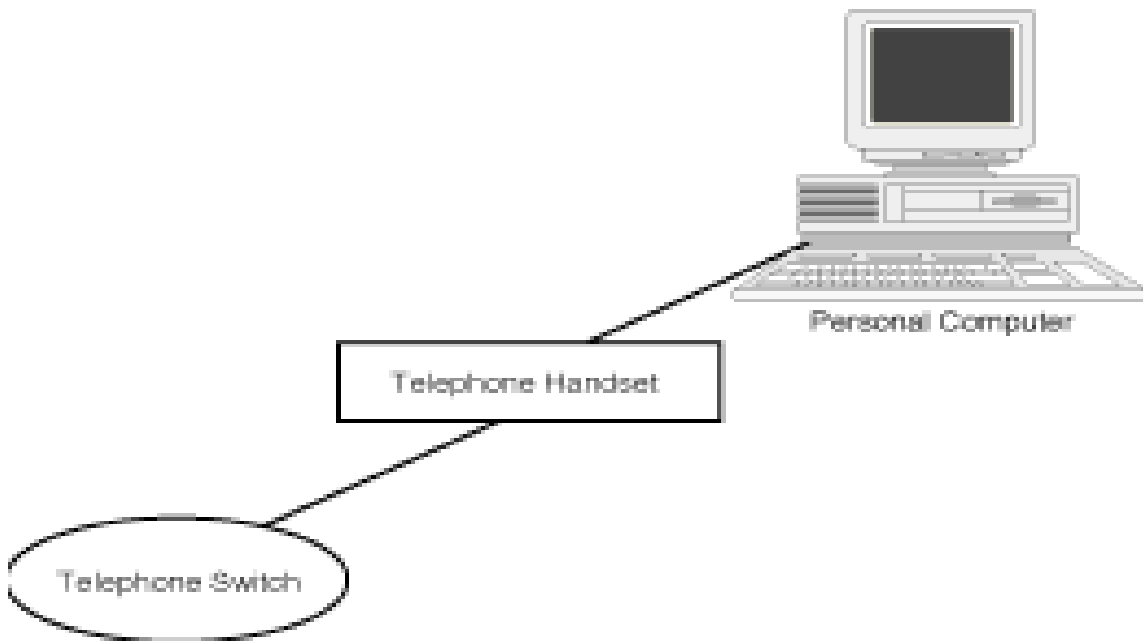


Figure 2: A typical phone-based TAPI configuration

This configuration is most useful when the telephone handset is the primary device for accessing the telephone.

4.2 PC-based configurations

PC-based TAPI configurations place the PC between the telephone switch and the standard handset. (see Figure 3).

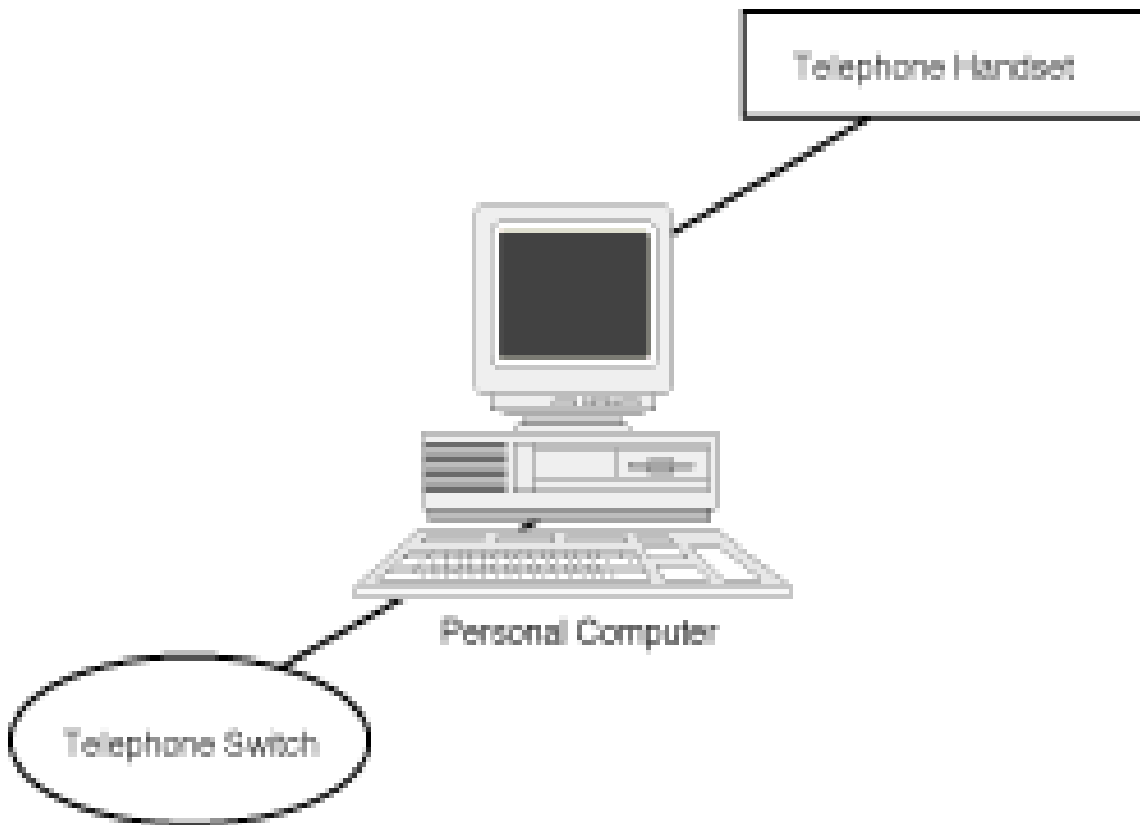


Figure 3: Typical pc-based TAPI configuration

This configuration is most useful when the PC is the primary device for accessing the telephone line. Another major advantage of the PC-based configuration is that the PC can act as a call manager for the handset. This is especially valuable in a mixed-mode environment where voice, data, and fax are all coming in to the same phone address.

In a PC-based configuration, the PC can also be used for call screening and message handling. TAPI-compliant software could record incoming messages for the user and place them in a queue for later review, or forward calls to another address. With the addition of caller ID services from the local telephone company, the PC could also act as a call filter agent, screening the calls as they arrive and allowing only designated callers access to the PC or the handset.

4.3 Shared or unified line configurations

The shared or unified line configuration is a bit of a compromise between PC-based and phone-based configurations. The shared line configuration involves a split along the line leading to the switch. Both the PC and the phone have equal (and simultaneous) access to the line (see Figure 4).

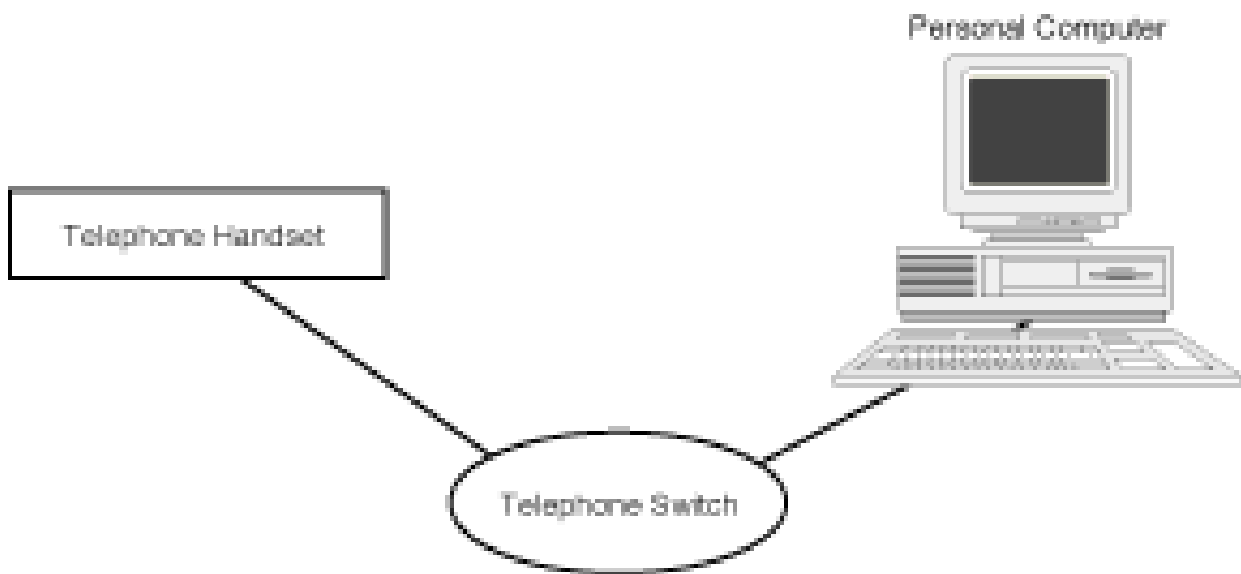


Figure 4: Typical shared line TAPI configuration

The advantage of the shared-line configuration is that either device can act as the originator of a call. All that is really needed is a microphone for input and speakers for output, but some systems offer headphones or a keypad to simulate the familiar telephone

handset. Also, with the unified arrangement, users do not need to worry about two devices ringing at the same time when a call comes in.

4.4 *Multi-line configurations*

So far, all the configurations reviewed here have been single-line models. This is commonly referred to as *first-party call control*. TAPI is also designed to support multi-line configurations. In this arrangement, TAPI is used to provide *third-party call control*. Single lines act as the first (and only) party in a telephone call. In a multi-line environment, a device can act as a third party in a telephone call. The most common form of third-party call control is a central switchboard in an office. When a call comes in, the switchboard (the third party) accepts the call, determines the final destination of the call, routes the call to the correct extension, and then drops out of the call stream.

These are the two basic multi-line TAPI configurations:

- **Voice server.** Used to provide voice mail and other services from a central location.
- **PBX server.** Used to provide call control for inbound and outbound trunk lines.

In a voice server configuration, the TAPI-enabled PC acts as a message storage device accessible from any other telephone handset.

In a PBX server configuration, the TAPI-enabled PC acts as a sort of first line of defense for all incoming calls to a multi-line location, usually an office building. In this mode, TAPI functions are used to accept calls, present them to an operator for review, and then forward them to the final destination. This is using the TAPI PC as a true third-party control system.

While a Voice Server does its work behind a standard desktop phone, the PBX Server does its work in front of any desktop phone. In other words, the Voice Server is designed to handle calls when the desktop phone is busy or in some other way unable to accept calls. The PBX Server, on the other hand, is used to accept all calls coming into the office and route those calls to the appropriate desktop phone. Many offices employ both PBX Server and Voice Server systems. The PBX Server answers the incoming line

and routes it to the desktop phone. If the desktop phone is unable to accept the call, the Voice Server takes a message and stores it for later retrieval.

5. Telephone line services

One of the primary aims of the TAPI model is to allow programmers to design systems that will work the same way regardless of the physical characteristics of the telephone line. TAPI functions behave the same on analog, digital, and cellular phone lines. TAPI is able to operate in single-line or multi-line configurations. In fact, the real value of the TAPI model for the programmer is that users can usually install TAPI-compliant software on systems with different physical line types and still operate properly.

It is important to know that some physical line types offer options not available on other line types. For example, ISDN lines offer simultaneous data and voice channels not available on POTS or T1 lines. TAPI cannot make a POTS line offer the same services an ISDN line offers. TAPI does, however, provide a consistent interface to all services options shared by line types (such as line open, dial, send data, close line, and so on).

Line types can be divided into three main groups:-

- Analog lines
- Digital lines
- Private protocol lines

Analog lines are the kind of lines available in most homes. Digital lines are usually used by large organizations, including local telephone service providers, to transfer large amounts of voice and data channels. T1 and ISDN lines are typical types of digital lines. Private protocol lines are a special kind of digital line. These lines are used within private branch exchanges (PBXs). PBX-type lines are used to transport voice, data, and special control information used by the switching hardware to provide advanced telephony features such as call transfer, conferencing, and so on.

5.1 Plain Old Telephone Service (POTS)

Plain Old Telephone Service (POTS) is the line service type provided to most homes. POTS is an analog service that provides basic connection to the telephone company's central office by way of a single-line link. Standard POTS users cannot perform advanced telephone operations such as call transfers, forwarding, or conferencing.

The analog POTS is designed to send voice signals, not data. For this reason, POTS users must employ a data modulator-demodulator (or *modem*) to send digital information over POTS lines.

5.2 Digital T1 lines

Digital T1 lines are designed to transport several conversations at once. T1 lines can send 24 multiple phone connections at the same time. Since T1 lines are digital instead of analog, data rates can extend well beyond the limits of analog lines. Data rates in the megabytes-per-minute are typical for dedicated T1 lines. T1 lines are typically used for dedicated data transmission and for bulk transmission of multiple conversations from point to point.

5.3 Integrated Services Digital Network (ISDN)

The *Integrated Services Digital Network (ISDN)* was developed to handle voice, data, and video services over the same line. The most common form of ISDN service, called *Basic Rate Interface* or *BRI-ISDN*, provides two 64Kbps channels and one 16Kbps *control* or *signal information* channel. The advantage of ISDN is that the two 64Kbps B channels can be configured to provide a single 128Kbps pipe for voice, video, or data, or the line can be configured to provide one 64Kbps data channel and a simultaneous 64Kbps digital voice channel. Thus ISDN service provides for transporting more than one media mode at the same time.

An advanced ISDN format, called *Primary Rate Interface* or *PRI-ISDN*, can provide up to 32 separate channels. PRI-ISDN, however, is a much more expensive service and is used by businesses that need to quickly move large amounts of data.

5.4 *Private Branch Exchange (PBX)*

The *Private Branch Exchange (PBX)* format is used in commercial switching equipment to handle multi-line phone systems in offices. Typically, the local telephone provider brings multi-line service up to the PBX switch and then the PBX handles all call control and line switching within the PBX network. PBX phones operate on proprietary formats and are usually digital-only lines.

6. TAPI architecture

Four different levels of TAPI services:-

- *Assisted telephony.* This is the simplest form of TAPI service.
- *Basic telephony.* This provides basic in- and outbound telephony services for a single-line phone.
- *Supplemental telephony.* This provides advanced telephone services such as hold, park, conference, and so on, to single and multi-line phones.
- *Extended telephony.* This provides a direct interface between Windows programs and vendor-specific TAPI services.

6.1 *Assisted telephony services*

<i>API Call</i>	<i>Parameters</i>	<i>Comments</i>
TapiRequestMakeCall	DestAddress, AppName, CalledParty, Comment	Use this function to request an outbound call placement. Only the DestAddress is required.
TapiGetLocationInfo	CountryCode, CityCode	Use this function to return the current country code and city code of the workstation. These values are stored in the TELEPHON.INI file.

Table 1: The assisted telephony API

The `TapiRequestMakeCall` has four parameters. Only the `DestAddress` is required. The `DestAddress` is a string of numbers that represents the telephone number to dial. The `AppName` parameter is the name of the application that requested the TAPI service. This would be the name of your application. The `CalledParty` is a string that represents the name of the person you are calling. The `Comment` parameter could contain a string of text describing the reason for the call.

The `TapiGetLocation` function returns two parameters: the `CountryCode` and `CityCode`.

6.2 Basic telephony services

Basic Telephony is the next level up in the TAPI service model. Basic Telephony function calls allow programmers to create applications that can provide basic in- and outbound voice and data calls over a single-line analog telephone. The analog phone line most often used for this level of service is often referred to as a *POTS* or *Plain Old Telephone Service* line. The Basic Telephony API set can also be used with more sophisticated lines such as T1, ISDN, or digital lines. However, the added features of these advanced line devices (such as call forwarding, park, hold, conference, and so on) are not available when using the Basic Telephony API set.

TAPI can "see" multiple TAPI devices on the same machine (data modem, handset, and fax board) while there is only one physical telephone line attached to the workstation.

6.2.1 The basic telephony line device API set

The Basic Telephony service model has several API calls handling and fulfilling service requests. These calls can be collected into logical groups:-

- ***Basic line-handling calls*** handle the initialization and opening and closing of TAPI lines.

- ***Line settings and status calls*** handle the reading and writing of various parameter values that control the behavior of the line device.
- ***Outbound and inbound functions*** handle the details of placing an outbound voice or data call and answering an inbound voice or data call.
- ***Outbound and inbound functions*** handle the details of recognizing, translating, and/or building telephone "addresses" or dialing strings.
- ***Miscellaneous features*** handle other TAPI-related functions, such as managing call-monitoring privileges and manipulating call handles.

6.3 Supplemental telephony services

The Supplemental telephony functions provide advanced line device handling (conference, park, hold, forward etc).

6.3.1 Supplemental telephony API for line devices

The Supplemental API set for line devices adds advanced call control and other features to the API library. The set can be divided into the following related groups of functions:

- ***Digit and tone handling functions*** allow programmers to detect and generate digits or tones along the phone line. This capability is needed to allow some systems to perform advanced line operations such as forwarding, call holds, and so on.
- ***Advanced line-handling functions*** provide call acceptance, rejection, redirection, and other operations. These are most useful in an environment where the phone line is connected to a central switch instead of directly to the external telephone service provider.
- ***Advanced call features functions*** provide Call Hold, Transfer, Park, Forward, and Pickup capabilities. These functions only work if the telephone line supports the advanced call features.

- ***Miscellaneous advanced features functions*** provide added features specific to TAPI service requests, such as monitoring lines and setting call parameters.

6.3.2 Supplemental telephony API for phone devices

The Supplemental Telephony API also provides function calls for the handling of phone devices. To TAPI, any device that can place or accept calls can be a phone device. The phone device API set allows programmers to invent their own phone devices in code. In effect, you can create a virtual handset using the TAPI phone device. This allows properly equipped workstations to act as single or multiple-line phones in an office environment. If our PC has appropriate audio input and output hardware (speakers, sound card, microphone, and so on) and is connected to the telephone service, we can create a "handset" using the phone device API set.

The Supplemental Telephony API set for phone devices can be divided into the following function groups:-

- ***Basic phone-handling functions*** provide basic initialization and shutdown, opening and closing a phone device, and ringing the open device.
- ***Phone settings and status functions*** allow programmers to read and write various settings of the phone device such as volume, gain, hookswitch behavior, and so on.
- ***Physical display, data, button, and lamp functions*** can be used to read and write display information to desktop units. Since TAPI can be used to support more than just PC workstations, these functions allow a central TAPI program to monitor and update LCD displays, to flash lamps, to change buttons labels, and to store and retrieve data from desktop terminals.

6.4 *Extended telephony services*

The last level of Telephony services is Extended Telephony. Extended Telephony service allows hardware vendors to define their own device-specific functions and services and still operate under the TAPI service model.

7. TAPI hardware considerations

Now we will see the differences between the three primary types of telephony hardware for PCs:-

- Basic data modems
- Voice-data modems
- Telephony cards

These three types of interface cards provide a wide range of telephony service for desktop workstations. We will see the advantages and limits of each of the interface card types and how we can use them in our telephony applications.

7.1 *Basic data modems*

These modems can support assisted telephony services (outbound dialing) and usually are able to support only limited inbound call handling.

7.2 *Voice-data modems*

These modems are a new breed of low-cost modems that provide additional features that come close to that of the higher-priced telephony cards. These modems usually are capable of supporting the Basic Telephony services and many of the Supplemental services. The key to success with voice-data modems is getting a good service provider interface for your card.

7.3 *Telephony cards*

These cards offer the greatest level of service compatibility. Telephony cards usually support all of the Basic Telephony and all of the Supplemental Telephony services, including phone device control. Most telephony cards also offer multiple lines

on a single card. This makes them ideal for supporting commercial-grade telephony applications.

Chapter 6

SPEECH APPLICATION PROGRAMMING INTERFACE

1. Introduction

Microsoft's Speech API or SAPI is an interface specification that enables developers to create applications for a variety of Speech Recognition (SR) and Text-to-Speech (TTS) engines from different vendors. It removes the need for developers to rewrite their applications, if and when they decide to change the underlying engines. Hence, SAPI is a popular choice over proprietary APIs. To give an analogy, it is what ODBC is to databases.

SAPI is a COM-based API. Voice applications consist of broadly two parts - Speech Recognition, and Text-to-Speech Synthesis. SAPI supports this functionality via its ISpRecognizer and ISpVoice interfaces. Speech Recognition occurs in two modes - Dictation, and Command & Control. Dictation assumes a set of all the words in a language. For example, Dictation mode in English would require the entire English dictionary. This mode requires a large amount of processing. Also, the voice patterns for a different user speaking the same set of words may differ. This necessitates the need of training the Speech engine to each user's voice patterns. In Command & Control mode the user's voice is recognized against an application-defined grammar, called CFG (Context Free Grammar). This grammar set contains a set of words which are expected to be spoken by the user, thus reducing the processing requirements for the Speech Engine. SAPI defines an XML format in which the application grammar can be defined. It can then be used via SAPI interfaces. Any application developer can design these CFGs and, if the subset of words expected from the user is small, forego the use of custom dictionaries. CFGs are also important tools for designing interactive Speech applications. They enable designing of intuitive prompts and recognition grammars.

2. The model/architecture

SAPI is a specification for both Engine and Application developers. Speech engine developers can make their engines SAPI compliant by implementing the Engine level interfaces. Application developers use the Application level interfaces provided by SAPI to access the features of the underlying engine. This means that the engine being used by an application can be changed without breaking application code. SAPI exposes

a wide set of interface to the application developers that covers most of the features provided by today's Speech engines. Additionally, it also allows engines to expose engine specific functionality via its interfaces. As mentioned, SAPI exposes interfaces for SR and TTS. We'll look at each of them.

Application

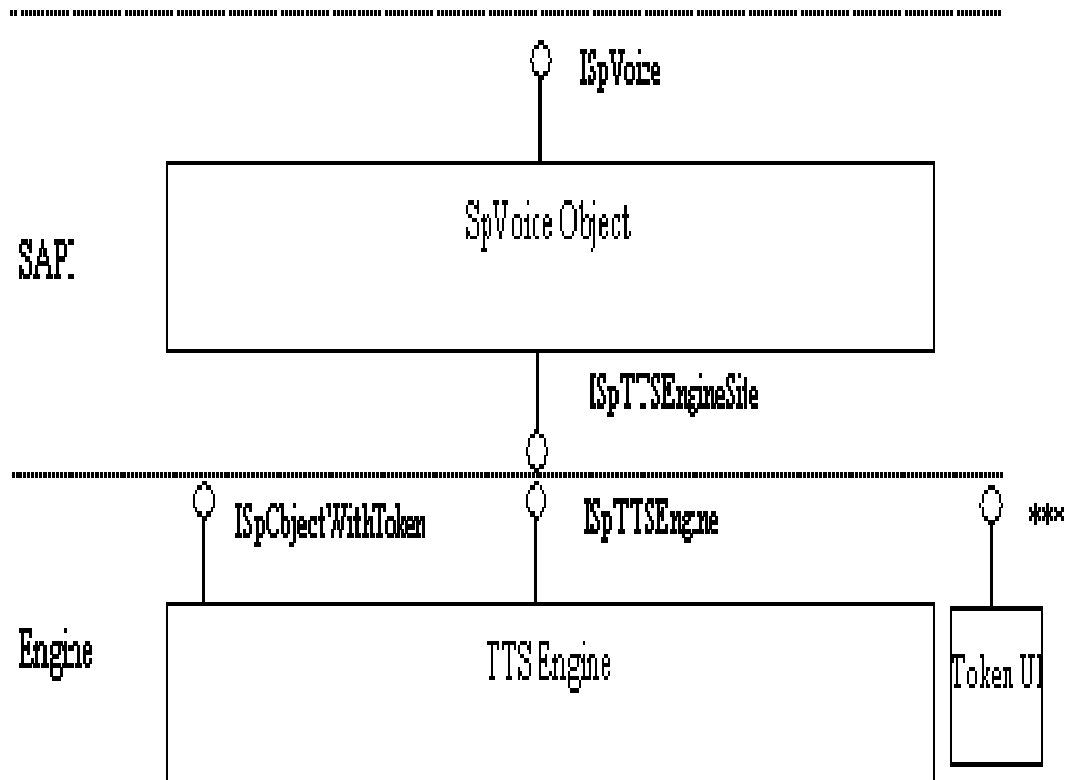


Figure 1: SAPI architecture

2.1 Automatic speech recognition

Each application uses a Recognizer object. This encapsulates the Recognition engine. Each Recognizer object can have multiple Recognition contexts on it. Speech recognition is performed by the engine via one or more recognition contexts. Applications can use multiple contexts for recognition pertaining to different parts of the application, or multiple applications, as we shall see. Let's say that one window requires

commands to invoke menu options, and another uses dictation for entering notes. Each of these windows can use a Recognition context that can be enabled as the specific window gets user focus.

Multiple grammar objects can be loaded on each Recognition context. Different parts of a window might need different grammars. For example, for a voice enabled Microsoft Windows Explorer, the grammar for folder browser would be different, enabling user to collapse or expand folders, or go to a specific drive. On the other hand, the file viewer may require commands like, "Open File..." or "Show Properties...". The application can query each recognition received on the context for the Grammar. This helps the application to take appropriate actions based on the recognition.

Further, each grammar object can allow loading of a CFG and a dictation topic simultaneously, or individually. For command-based applications, a CFG can be loaded from a file. This CFG conforms to the SAPI Grammar syntax. For dictation, a dictation topic can be loaded. This is essentially a domain specific dictionary containing words specific to let's say, medicine ("Anesthesia, Cortisone, Paracetamol...."), or scientific ("Quantization, Quasar..."). These dictionaries are provided by engine vendors. Vendors also provide a default dictionary for a language. This is useful in taking notes, or Dictaphone type of applications. The combination of both is useful, especially in a word processing application where both commands and dictation are needed. In such a case, the user's speech will first be recognized against the CFG, and if recognition fails, dictation will kick in.

The Recognizer is available in two forms:-

- Shared
- InProc

2.1.1 Shared

Recognizer is useful in desktop applications where multiple applications want to use a common system wide instance of a Recognizer, residing in a separate process. Applications create their Reco contexts on the common recognizer. But the limitation with this is that the speech input to the recognizer can be a through a common device,

such as the system microphone. In fact, when the recognizer is first initialized as shared, SAPI sets up the microphone as the default input device.

2.1.2 InProc recognizer

Each InProc instance is created for its calling application. The input stream needs to be set manually in this case. This can be set as wave file, microphone, or raw wave data. InProc, recognizers are used with telephony or server applications, where each application would need complete control over the recognizer, for activation, deactivation, setting input streams, etc. SAPI supports multiple wave formats. They cover the most common PCM, TrueSpeech, A-Law, μ -Law, ADPCM, and GSM formats.

2.2 Text-to-Speech

SAPI exposes TTS functionality using the following model: SAPI exposes a single interface for its TTS functionality, ISpVoice. The engine vendor supplies its TTS engine with a set of one or more voices. These voices can be characterized using a set of attributes such as Male/Female, Adult/Child, and language supported by the voice. When created, the ISpVoice object is ready to use with the default voice specified under the Control Panel's Speech settings dialog. Once created, however, the voices can be changed. SAPI provides methods to query for available voices on the engine. Voices can be set depending on the requirements of the application. Some may require an adult male voice. Others may need a female child voice. Each voice has a characteristic vocalization, which may or may not provide the best user experience in certain situations.

SAPI provides application control over many parameters that can alter the vocalization of each voice. Some of these can be controlled using the ISpVoice methods. These include, pitch, volume and rate adjustment. Others can be controlled using XML TTS markup tags. SAPI provides control over voice, silence, emphasis and the grammatical emphasis of words in addition to pitch, volume and rate. These tags are embedded in the text that needs to be synthesized.

2.2.1 Possible applications for Text-to-Speech

TTS has the following practical uses:-

- **Reading dynamic text.** TTS is useful for phrases that vary too much to record and store using all possible alternatives.
- **Proofreading.** Audible proofreading of text can help users catch errors missed by visual proofreading.
- **Conserving storage space.** TTS is useful for phrases that would take up too much storage space if they were prerecorded in a digital-audio format.
- **Event notification and audible feedback.** TTS works well for informational messages. For example, a cashier is entering a sale into a cash register and enters an obviously unlikely quantity. A TTS-enabled application could inform the cashier of the mistake and proceed or cancel it depending on the cashier's reaction. This type of notification should only be used in conjunction with a visible message in case the user turns the sound off or is out of hearing range.
- **Telephony.** One of the major up-and-coming uses for TTS is in the area of telephony, the use of computers and telephones. Imagine an answering machine that could tailor its outgoing answer depending on the caller ID.

3. SAPI's strengths

Following are SAPI's strengths as discussed in various sections above:-

- **Standard Specification** Adoption enables compliance with several speech engines, without additional development effort.
- **COM based API** Supports application development and scripting. No learning curve in terms of implementation.
- **Oriented towards feature** Rich desktop applications.

4. SAPI's Weaknesses

Following are SAPI's weaknesses:-

- ***Limited to Microsoft Platform.***
- ***No direct server-side implementation*** Default implementation limited to a single user deployment scenario. The interfaces are not designed for large portal development with multiple access points.
- ***Engine specific profiles*** User Profiles can't be ported across speech engines, as SAPI does not define standards for profile storage. Thus, Training required for each user on each engine.

Chapter 7

WIRELESS ACCESS PROTOCOL

1. Introduction

The Wireless Application Protocol (WAP) is the de-facto world standard for the presentation and delivery of wireless information and telephony services on mobile phones and other wireless terminals. WAP allows carriers to strengthen their service offerings by providing subscribers with the information they want and need while on the move. Enabling information access from handheld devices requires a deep understanding of both technical and market issues that are unique to the wireless environment.

The WAP specification was developed by the industry's best minds to address these issues. Wireless devices represent the ultimate constrained computing device with limited CPU, memory, and battery life, and a simple user interface. Wireless networks are constrained by low bandwidth, high latency, and unpredictable availability and stability. However, most important of all, wireless subscribers have a different set of essential desires and needs than desktop or even laptop Internet users. WAP-enabled devices are companion products that will deliver timely information and accept transactions and inquiries when the user is moving around. WAP services provide pinpoint information access and delivery when the full screen environment is either not available or not necessary. The WAP specification addresses these issues by using the best of existing standards, and developing new extensions where needed. It enables industry participants to develop solutions that are air interface independent, device independent and fully interoperable.

WAP is published by the WAP Forum, founded in 1997 by Ericsson, Motorola, Nokia, and Unwired Planet. Forum members now represent over 90% of the global handset market, as well as leading infrastructure providers, software developers and other organizations.

2. Why WAP is necessary?

2.1 Ensure interoperability

Service providers must feel secure that their investments will yield benefits in the future. They will not be able to do so until equipment and software offered by different suppliers can be made to work together. The WAP specification has been designed to encourage easy, open interoperability between its key components. Any solution

component built to be compliant with the WAP specification can interoperate with any other WAP-compliant component.

Bearer and device independence both help foster interoperability. But interoperability goes beyond these two principles to require that each WAP compatible component will communicate with all other components in the solution network by using the standard methods and protocols defined in the specification. Interoperability provides clear benefits for handset manufacturers and infrastructure providers. Handset manufacturers are assured that if their device complies with the WAP specification it will be able to interface with any WAP-compliant server, regardless of the manufacturer. Likewise, the makers of a WAP-compliant server are assured that any WAP-compliant handset will interface correctly with their servers.

2.2 Encourage and foster market development

The WAP specification is designed to bring Internet access to the wireless mass market. By building open specifications, and encouraging communication and technical exchanges among the industry players, the WAP Forum has already begun to open the wireless data market in new ways. Just over a year ago, the idea of a single wireless data standard was unheard of, yet today the WAP specification is available to the public, and dozens of companies are promoting this vision of the future. The revolution is under way to bring information access to any handset, at a reasonable price and in an easy to use form factor. Providing Internet and Web-based services on a wireless data network presents many challenges to wireless service providers, application developers and handset manufacturers. While the obvious limitations are rooted in the nature of wireless devices and data networks, there are also more fundamental differences that are important to understand.

2.2.1 The market is different

Bringing computing power to a wireless handset opens an extensive new market for information access. This market is very different from the traditional desktop or even the laptop market because the subscriber has a different set of needs and expectations. Some of these differences include:-

- ***Ease of use.*** Despite the fact that using a desktop computer has become progressively easier over the last five years, a wireless computing device must be dramatically easier to use than even the simplest desktop computer. These devices will be used by people who potentially have no desktop computing experience. Furthermore, they will often be used in a dynamic environment where the user is engaged in multiple activities. Subscribers won't be focused on their handset the way they are when they are sitting in front of a desktop computer. Therefore, the devices must be extremely simple and easy to use. Applications built for these devices must therefore present the best possible user interface for quick and simple usage. There can be no installation scripts, complicated menu structure, application errors, general protection faults or complicated key sequences such as ctrl-alt-del, or alt-shift-F5.
- ***Market size.*** The growth and size of the wireless subscriber market has been phenomenal. According to Global Mobile magazine, there are more than 200 million wireless subscribers in the world today. According to Nokia, there will be more than one billion wireless subscribers by the year 2005. The wireless market is enormous: it can afford and will demand optimized solutions.
- ***Price sensitivity.*** Even with today's sub-\$1000 computers, a price difference of \$50 between two models is not considered significant. However, a difference of \$50 between two handsets is very significant, especially after years of subsidized handset pricing by the service provider. Market studies have shown that a mass-market handset must be priced under \$149 to be competitive. A solution must add significant value at a low cost to be effective in this market.
- ***Usage patterns.*** Subscribers expect wireless data access to perform like the rest of their handset: The service should be instantly available, easy to use and designed to be used for a few minutes at a time. Hourglass icons telling subscribers to wait will not be acceptable.

- *Essential tasks.* As soon as professionals step out of the office, information needs and desires change. Wireless Internet subscribers do not want to use their handset to "surf the Internet." They have small, specific tasks that need to be accomplished quickly. Subscribers want to scan email rather than read it all, or see just the top stock quotes of interest. Receiving timely traffic alerts on the handset is essential, whereas the same information may not be as valuable at the desktop.

2.2.2 The network is different

Wireless data networks present a more constrained communication environment compared to wired networks. Because of fundamental limitations of power, available spectrum and mobility, wireless data networks tend to have:-

- Less bandwidth
- More latency
- Less connection stability
- Less predictable availability

Furthermore, as bandwidth increases, the handset's power consumption also increases which further taxes the already limited battery life of a mobile device. Therefore, even as wireless networks capitalize on higher bandwidth, the power of a handset is always limited by battery capacity and size, thus challenging the amount of data throughput. Deployment of the WAP standard accommodates more users per MHz since it uses the available bandwidth at an extremely efficient level. The result of placing more users on a given amount of spectrum can yield lower costs for both the network provider and the customer. A wireless data solution must be able to overcome these network limitations and still deliver a satisfactory user experience.

2.2.3 The device is different

Similarly, mass-market, handheld wireless devices present a more constrained computing environment compared to desktop computers. Because of fundamental limitations of battery life and form factor, mass-market handheld devices tend to have:

- Less powerful CPUs
- Less memory (ROM and RAM)

- Restricted power consumption
- Smaller displays
- Different input devices (e.g., a phone keypad, voice input, etc.)

Because of these limitations, the user interface of a wireless handset is fundamentally different than that of a desktop computer. The limited screen size and lack of a mouse requires a different user interface metaphor than the traditional desktop GUI. These conditions are not likely to change dramatically in the near future. The most popular wireless handsets have been designed to be lightweight and fit comfortably in the palm of a hand. Furthermore, consumers desire handsets with longer battery life, which always limits available bandwidth, and the power consumption of the CPU, memory and display. Because there will always be a performance gap between the very best desktop computers and the very best handheld devices, the method used to deliver wireless data to these devices will have to effectively address this gap. As this gap changes over time, standards will have to continually evolve to keep pace with available functionality and market needs.

3. WAP specification

The WAP specification is a major achievement because it defines for the first time an open, standard architecture and set of protocols intended to implement wireless Internet access. It also provides solutions for problems not solved by other standardization bodies and is a catalyst for wireless development and standardization. The key elements of the WAP specification include:-

- ***WAP programming model.*** As seen in figure 1 which is based heavily on the existing WWW Programming Model. This provides several benefits to the application developer community, including a familiar programming model, a proven architecture and the ability to leverage existing tools (e.g., Web servers, XML tools, etc.). Optimizations and extensions have been made in order to match the characteristics of the wireless environment. Wherever possible, existing standards have been adopted or have been used as the starting point for WAP technology.

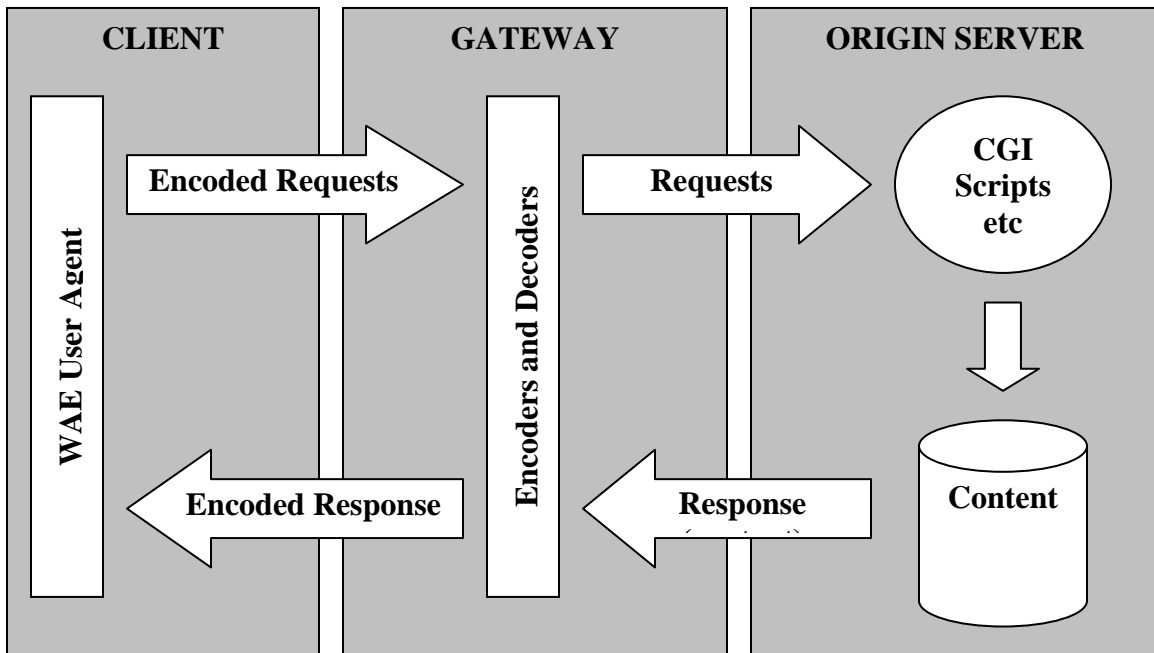


Figure 1: The WAP programming model

- Markup language.** The Wireless Markup Language (WML) and WML Script do not assume that a keyboard or a mouse are available for user input, and are designed for small screen displays. Unlike the flat structure of HTML documents, WML documents are divided into a set of well-defined units of user interactions. One unit of interaction is called a card, and services are created by letting the user navigate back and forth between cards from one or several WML documents. WML provides a smaller, telephony aware, set of markup tags that makes it more appropriate than HTML to implement within handheld devices. From the WAP Gateway, all WML content is accessed over the Internet using standard HTTP requests, so traditional Web servers, tools and techniques are used to server this new market.
- A specification for a microbrowser.** In the wireless terminal that controls the user interface and is analogous to a standard Web browser. This

specification defines how WML and WML Script should be interpreted in the handset and presented to the user. The microbrowser specification has been designed for wireless handsets so that the resulting code will be compact and efficient, yet provide a flexible and powerful user interface.

- *A framework for Wireless Telephony Applications (WTA)*. Allows access to telephony functionality such as call control, phone book access and messaging from within WML Script applets. This allows operators to develop secure telephony applications integrated into WML/WML Script services. For example, services such as Call Forwarding may provide a user interface that prompts the user to make a choice between accepting a call, forwarding it to another person or forwarding it to voicemail.

4. WAP solution benefits

The WAP specification was written to address the challenges of traditional wireless data access within the context of the design objectives of the WAP Forum. This section outlines how the WAP specification meets these goals.

4.1 Delivers an appropriate user experience model

The WAP specification defines a powerful and functional user interface model that is appropriate for handheld devices. Users navigate through cards with up and down scroll keys instead of a mouse. Soft keys allow the user to perform specific operations appropriate to the application context, or select menu options. A traditional 12-key phone keypad is used to enter alphanumeric characters, including a full set of standard symbols. Navigation functions such as Back, Home, and Bookmark are also provided, in keeping with the standard browser model. By using the existing Internet model as a starting point, this user interface provides familiar functionality for those accustomed with the Web. It also provides a user interface that is easy to learn and highly discoverable for the first time user. The microbrowser allows devices with larger screens and more features to automatically display more content, just as a traditional browser does on a PC when the browser window is expanded on screen.

4.2 Leverages proxy technology

The WAP specification uses standard Web proxy technology to connect the wireless domain with the Web. By using the computing resources in the WAP Gateway, the WAP architecture permits the handset to be simple and inexpensive. For example, a WAP Gateway will typically take over all DNS services to resolve domain names used in URLs, thus offloading this computing task from the handset. The WAP Gateway can also be used to provision services to subscribers and provide the network operator with a control point to manage fraud and service utilization. A WAP Gateway typically includes the following functionality:-

- ***Protocol gateway.*** The protocol gateway translates requests from the WAP protocol stack to the WWW protocol stack (HTTP and TCP/IP).
- ***Content encoders and decoders.*** The content encoders translate Web content into compact encoded formats to reduce the size and number of packets traveling over the wireless data network. This infrastructure ensures that mobile terminal users can browse a variety of WAP content and applications regardless of the wireless network they use. Application authors are able to build content services and applications that are network and terminal independent, allowing their applications to reach the largest possible audience. Because of the WAP proxy design, content and applications are hosted on standard WWW servers and can be developed using proven Web technologies such as CGI scripting.

The WAP Gateway decreases the response time to the handheld device by aggregating data from different servers on the Web, and caching frequently used information. The WAP Gateway can also interface with subscriber databases and use information from the wireless network, such as location information, to dynamically customize WML pages for a certain group of users.

4.3 Addresses the constraints of a wireless network

The protocol stack defined in WAP optimizes standard Web protocols, such as HTTP, for use under the low bandwidth, high latency conditions often found in wireless networks. A number of enhancements to the session, transaction, security and transport

layers provide HTTP functionality better suited to the wireless network environment. Here are just a few examples of these improvements:-

- The plain text headers of HTTP are translated into binary code that significantly reduces the amount of data that must be transmitted over the air interface.
- A lightweight session reestablishment protocol has been defined that allows sessions to be suspended and resumed without the overhead of initial establishment. This allows a session to be suspended while idle to free up network resources or save battery power.
- WAP provides a Wireless Transaction Protocol (WTP) that provides reliable transport for the WAP datagram service. WTP provides similar reliability as traditional TCP does, but without behaviors that make TCP unsuitable in a wireless network. For example, TCP transmits a large amount of information for each request-response transaction, including information needed to handle out of order packet delivery. Since there is only one possible route between the WAP proxy and the handset, there is no need to handle this situation. WTP eliminates this unnecessary information and reduces the amount of information needed for each request-response transaction. This is just one example of the optimizations WTP provides.
- WAP's WTP solution also means that a TCP stack is not required in the phone, which allows for significant savings in processing and memory cost in the handset.

4.4 Provides a secure wireless connection

Many applications on the Web today require a secure connection between the client and the application server. The WAP specification ensures that a secure protocol is available for these transactions on a wireless handset. The Wireless Transport Layer Security (WTLS) protocol is based upon the industry-standard Transport Layer Security (TLS) protocol, formerly known as Secure Sockets Layer (SSL). WTLS is intended for use with the WAP transport protocols and has been optimized for use over narrow-band

communication channels. WTLS ensures data integrity, privacy, authentication and denial-of-service protection. For Web applications that employ standard Internet security techniques with TLS, the WAP Gateway automatically and transparently manages wireless security with minimal overhead.

4.5 Optimized for handheld wireless devices

The WAP specification defines a microbrowser that is the ultimate thin client, able to fit in a limited amount of memory in the handheld device. The use of proxy technology and compression in the network interface reduces the processing load at the handheld device so that an inexpensive CPU can be used in the handset. This further helps reduce power consumption and extends battery life, meeting the needs of both handset manufacturers and wireless subscribers.

4.6 Implements new wireless functionality

The WAP specification also defines new functionality that has not been defined by any other standard, such as a voice/data integration API and the groundwork for wireless push functionality.

The Wireless Telephony Application (WTA) allows application developers to initiate phone calls from the browser and respond to network events as they occur. The WTA API accomplishes this by providing an interface to the local and network telephony infrastructure. The local interface allows WML and WML Script to access a specific set of telephony functions, such as a function call to dial a phone number from the mobile handset. The network interface allows an application to monitor and initiate mobile network events, so that the application can take action or update information based on these events. This functionality can be used to keep an updated list of the phone numbers dialed into an active conference call. These network and local APIs are powerful features that no other standard provides.

Standard HTTP has no support for "push" functionality. The WAP specification defines a push mechanism that will allow any Web server to send information to the client. This is an extremely important feature because it allows applications to alert the subscriber when time-sensitive information changes. There are a number of applications

that make use of this functionality, such as traffic alerts and stock quote triggers, or email and pager notifications.

4.7 Enables application development using existing tools

Web developers will find it easy to develop WAP applications since the WAP Programming model closely follows the existing WWW development model. WML is a tag based document language specified as an XML document type. As such, existing XML authoring tools, as well as many HTML development environments, can be used to develop WML applications. Since the WAP specification uses standard HTTP protocol to communicate between the WAP Gateway and Web servers, Web developers can deploy their applications on any off-the-shelf Web server. WML developers can use standard Web tools and mechanisms such as Cold Fusion, CGI, Perl, ASP and others to generate dynamic WML applications.

Developers can either use separate URLs for their HTML and WML entry points, or use a single URL to dynamically serve either HTML or WML content according to the requestor's browser type. Although it is possible to translate HTML into WML using an automated system, in practice the best applications use WML to tailor the interface to the specific needs of the wireless user. This allows for the best possible use of the handset features, such as soft keys, and provides the best user experience.

The most valuable parts of any Web application are typically the unique content it provides and the back-end database interaction, not the particular HTML that was written to interact with the user. Therefore developing a corresponding WML front-end leverages previous engineering effort, while providing significant user interface benefits.

4.8 Adapts new standards for the industry

Wherever possible, the WAP specification optimizes and extends existing Internet standards. The WAP Forum has taken technology elements from TCP/IP, HTTP and XML, optimized them for the wireless environment, and is now submitting these optimizations to the W3C standards process as input for the next generations of (XHTML) and HTTP (HTTP-NG). The WAP Forum will continue to evolve the WAP specification to keep pace with new technologies. In the best tradition of Internet protocol standards, the WAP specification divides network functionality into several layers, so

that each layer can develop independently of the others. Low level layers can be replaced to support new bearers without requiring changes to the high level APIs or the intervening stack layers. This protects the initial investment in the protocol stack, and makes the standard flexible as new and faster wireless data protocols become available.

5. How developers benefit from using WAP-based solutions?

Application developers can reach the largest possible audience when they write their applications in WML because they are writing to an industry standard. Additional benefits for developers include:-

- Access to an entirely new, immense market of information-hungry wireless subscribers, while complementing their existing Internet services.
- Because WML is an XML-based language, it is an easy markup language for existing Web developers to learn.
- WML's basis in XML also positions it well as a future target markup language for automatic content transformation. The W3C is currently defining the eXtensible Style Language (XSL), which provides a powerful mechanism for the dynamic transformation of well-formed XML. Using an XSL stylesheet, content written in XML-defined markup languages can be automatically translated into content suitable for either HTML or WML, as shown in Figure 2. Likewise, content written in well-formed XML can also be translated to other XML-based markup languages, using a different XSL style sheet.

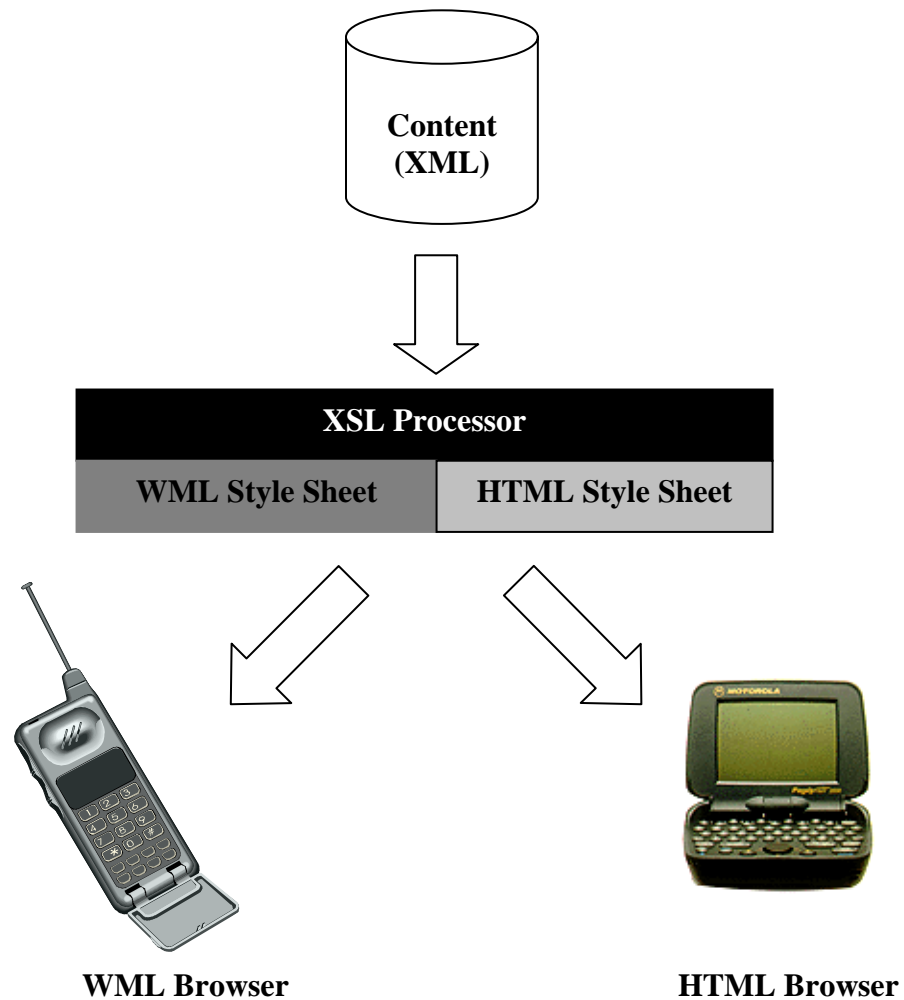


Figure 2: The future of content development

- While the technology for universal content is still being developed, WML has been designed to be an integral part of this technology. Application developers can feel secure using WML today, knowing that there will be a migration path to the future.
- Since WML is part of an open standard, and was developed by an independent organization, all developers can be assured that they are on equal footing with other developers. No single developer has unique access to APIs or special functionality.

- By writing in WML, a developer's work becomes available to any network and device that is WAP-compliant. WML and the WAP specification truly deliver on the "write once, use anywhere" promise.
- WML provides the application developer with the power to take full advantage of the user interface. Applications can map soft keys for easy user input and use special features to maximize the effect of displaying text on a limited screen.
- WML allows application developers to integrate their applications with device and network telephony functions. applications that use these features can truly leverage the advantages of operating in an integrated voice and data device.
- WML allows the use of icons and bitmapped graphics, for devices that support them. One application will work equally well on a phone with or without graphics by offering alternate text to the phone that is not capable of displaying images.
- An application written in WML will look good on any device that is WAP-compliant. If one device is able to display more lines of text than another, the microbrowser will do so automatically, making the best use of the device's and application's capabilities.
- An application can be customized to take advantage of a particular device's capabilities, by using standard HTTP header mechanisms to learn about the device's capabilities.

6. How subscribers benefit from using WAP-based solutions?

Ultimately, subscribers are the most important beneficiaries of the WAP standards. The WAP specification was developed and written by experienced telecommunications experts who not only understand the technologies involved, but also the real needs of the subscriber. Consequently, the WAP specification delivers significant value to the subscriber.

The WAP specification pulls together existing technologies and defines new standards to provide subscribers with:-

- Fast, efficient access to essential information from a wireless handset.
- Peace of mind that all transactions are completely secure.
- An easy to use interface metaphor that meets the needs of the user within the restrictions of a constrained network and device.

The widespread adoption of the WAP specification is yielding these benefits:-

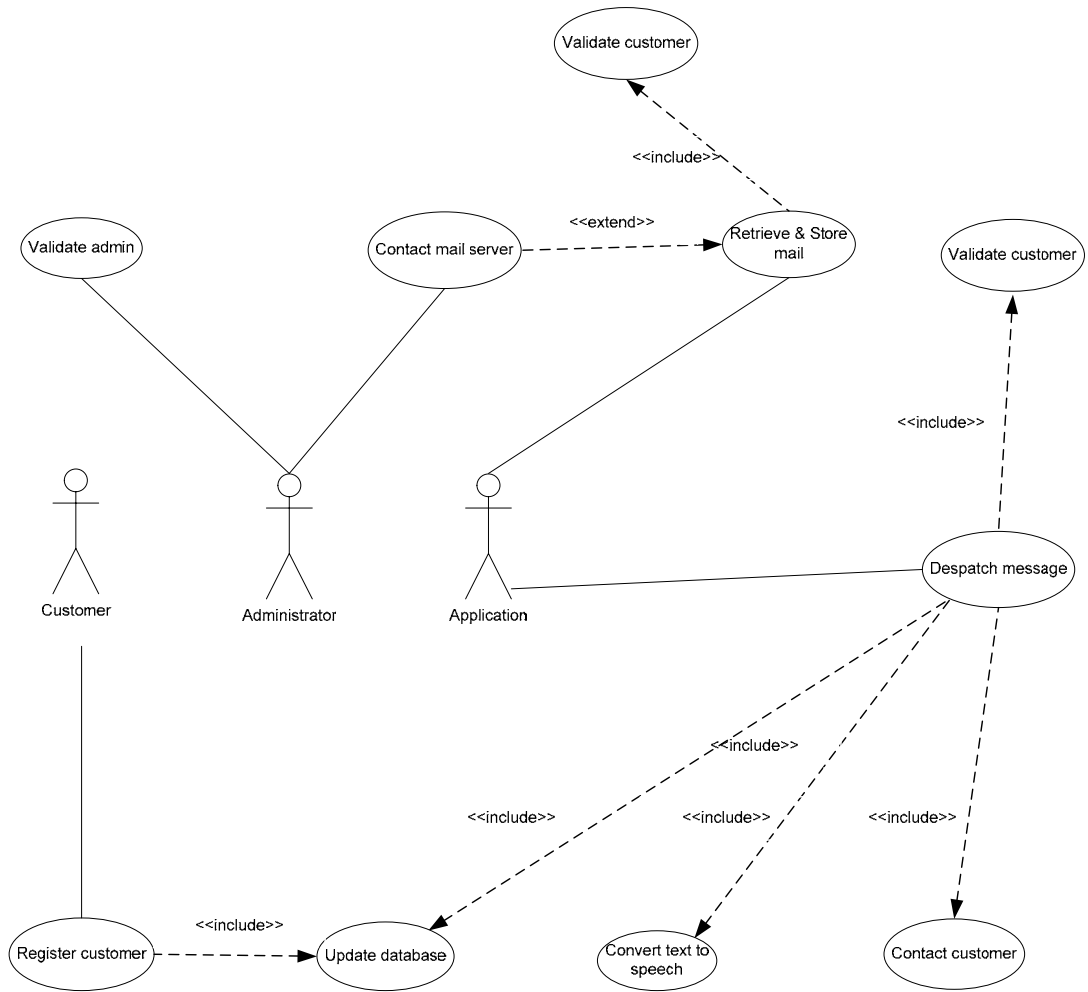
- ***A common user interface*** metaphor that is being used by all industry participants. Just as the desktop metaphor is the de-facto standard for applications on PCs, the WAP card metaphor provides a common interface to which all applications can conform.
- ***Ubiquity of service***. Wherever subscribers go, they will have access to their own personal content using a WAP-enabled browser.
- ***Wide selection of devices***. In addition to handsets with different features and form factors, subscribers will be able to use PDAs and pagers that are also WAP-enabled.
- ***A large selection of applications***. Over the last few years, the Internet model has proven to be the least expensive and most effective way to deliver new applications and services to computing users. Now that this model has been extended to wireless devices, subscribers will gain access to a wealth of applications.

Chapter 8

DESIGN AND IMPLEMENTATION

1. Use case diagram

Use Case Analysis



2. Use case description

No	Use case 1
Name	Validate admin
Actor	Administrator
Preconditions	Login screen is available to the administrator
Postconditions	Main screen is initialized.
Description	Enter user name Enter password. Press login Open connection Verify user name and password.. Initialize main screen

No	Use case 2
Name	Contact mail server
Actor	Administrator
Preconditions	All conditions required for internet connection be met
Postconditions	Contact with the POP3 compliant mail server is established.
Description	Open port Enter user name and password. Verify user name and password. Establish connection.

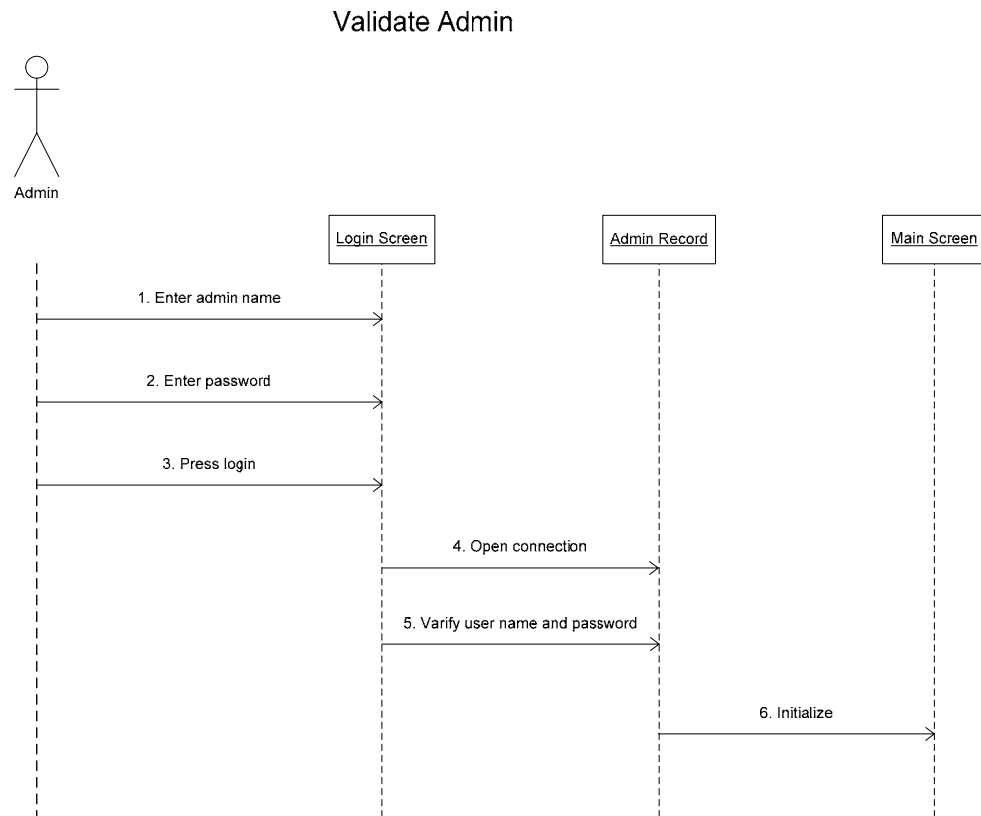
No	Use case 3
-----------	------------

Name	Retrieve and store mail
Actor	Application
Preconditions	Contact with the POP3 compliant mail server is established.
Postconditions	Mails of all the subscribers are retrieved and stored in the database.
Description:	Enter user ID. Verify user ID. Download mail. Establish connection with the database. Store mail.

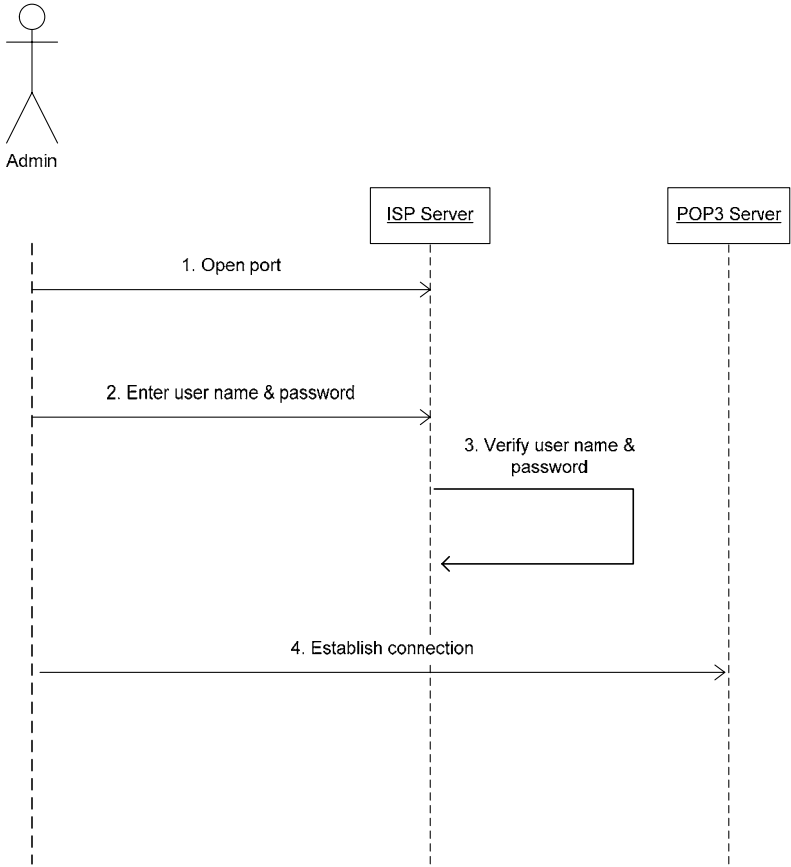
No	Use case 4
Name	Dispatch mail
Actor	Application
Postconditions	Mails of all the subscribers are delivered either in voice or textual form and the database is updated accordingly.
Description	Contact customer. Prompt for ID. Send ID. Verify ID. Prompt for choice. Enter choice. Retrieve mail from the database. Convert TTS. Send mail. Update delivery record.
Variations	If the user is a mobile phone user, he can view his mail in textual form. In that case tts conversion is bypassed

No	Use case 5
Name	Register customer
Actor	Customer
Postconditions	Customer is added to the system and the database is updated accordingly.
Description	Contact administrator. Send data. Process data. Update customer record.

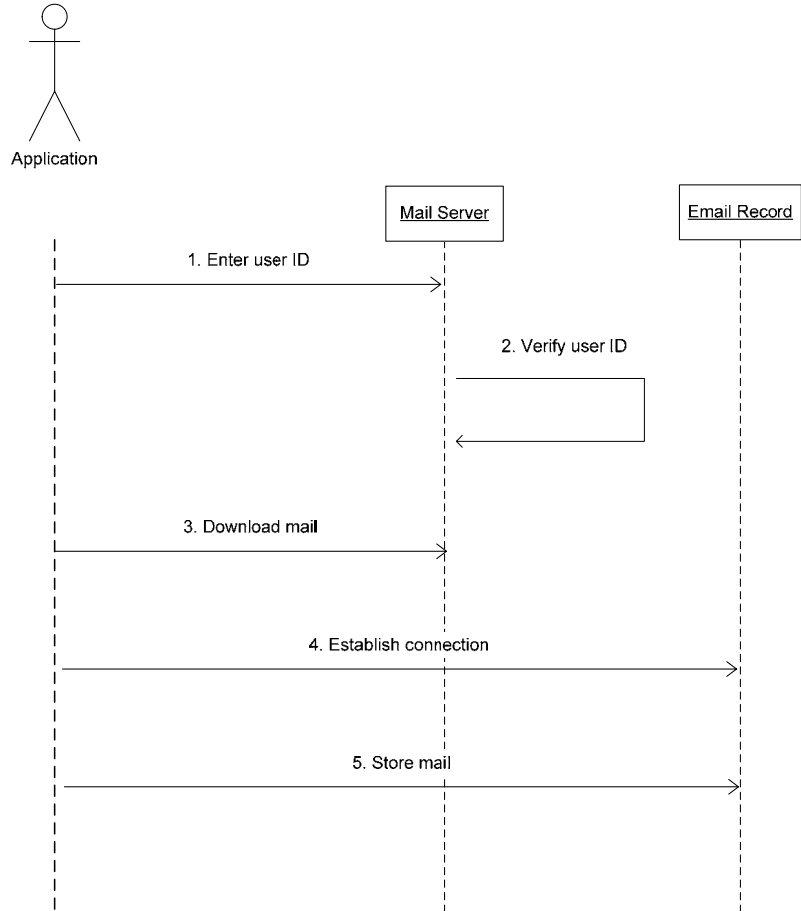
3. Sequence diagrams



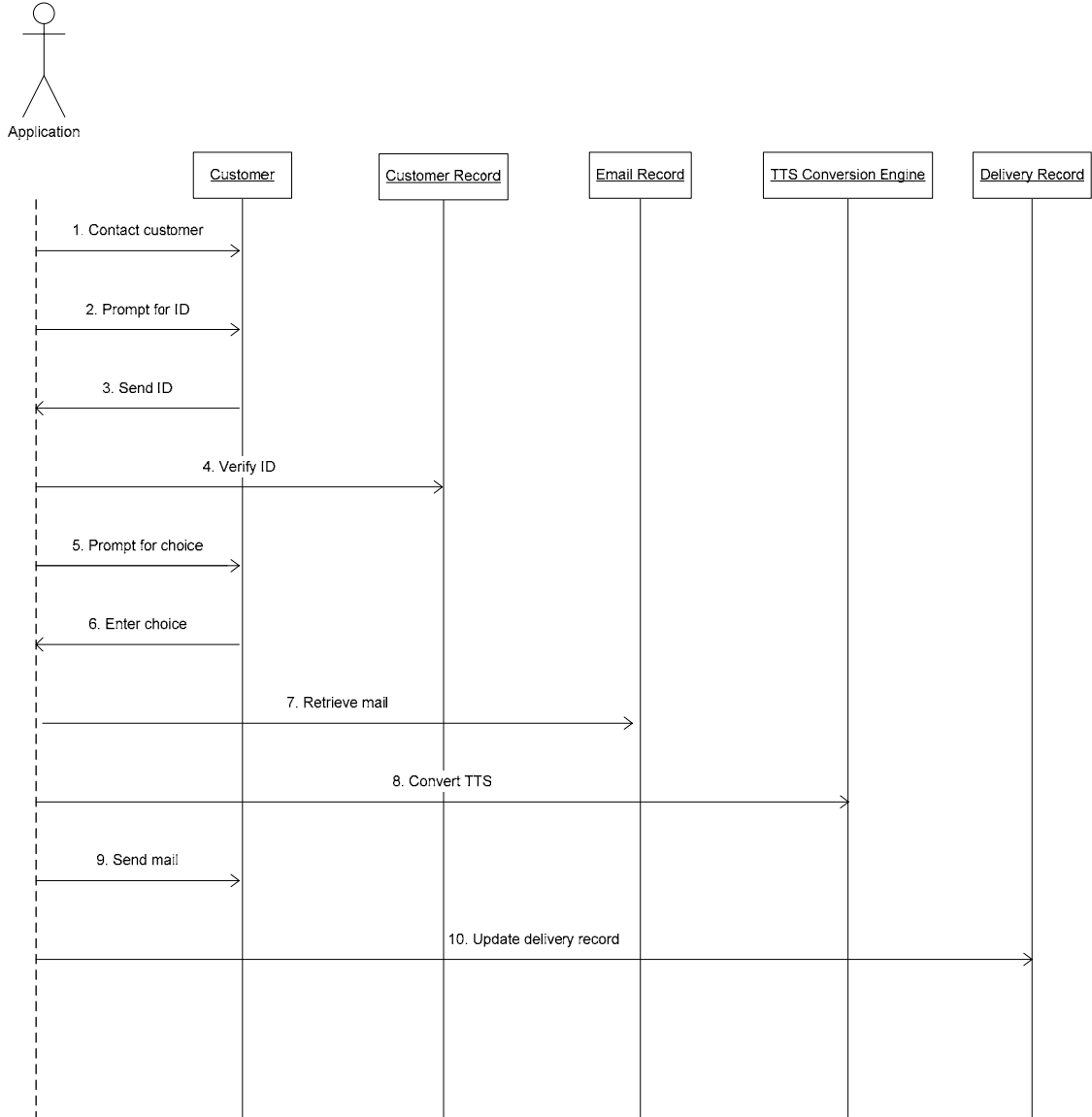
Contact Mail Server



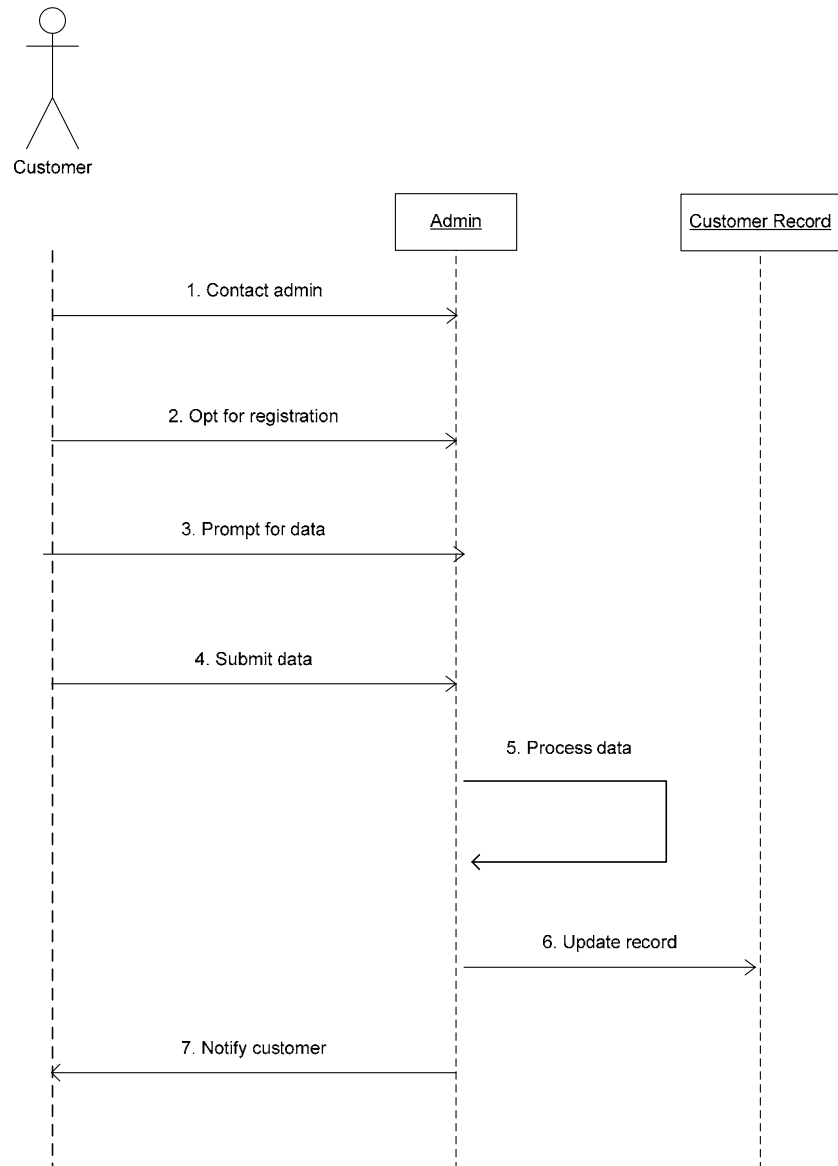
Retrieve & Store Mail



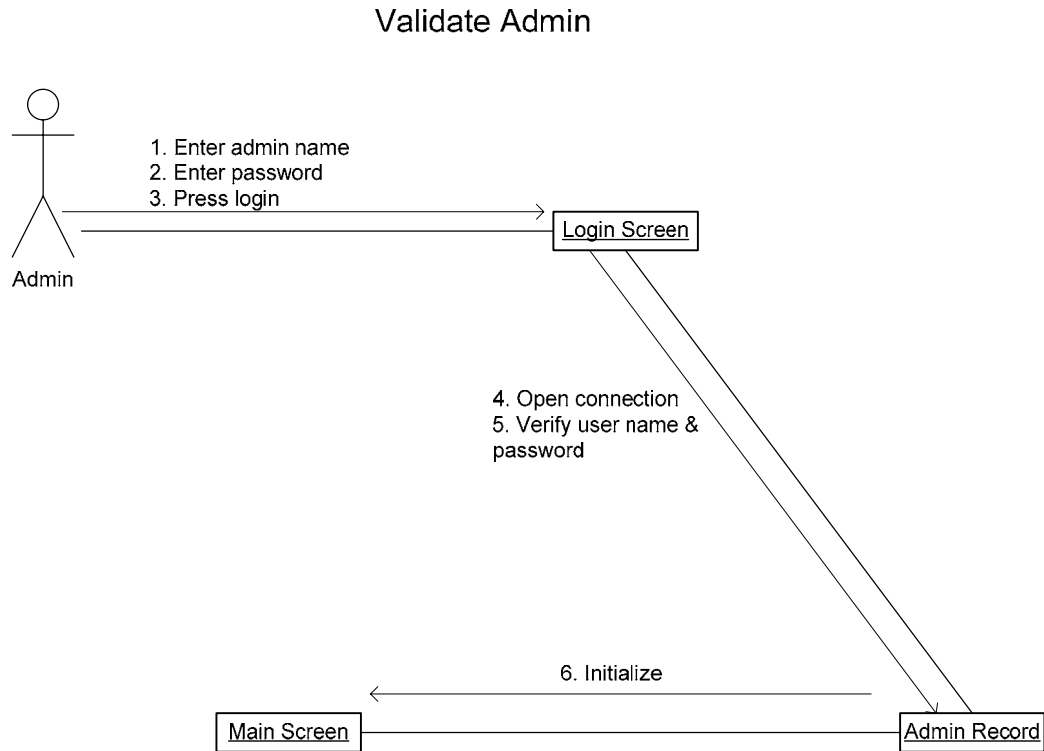
Dispatch Mail



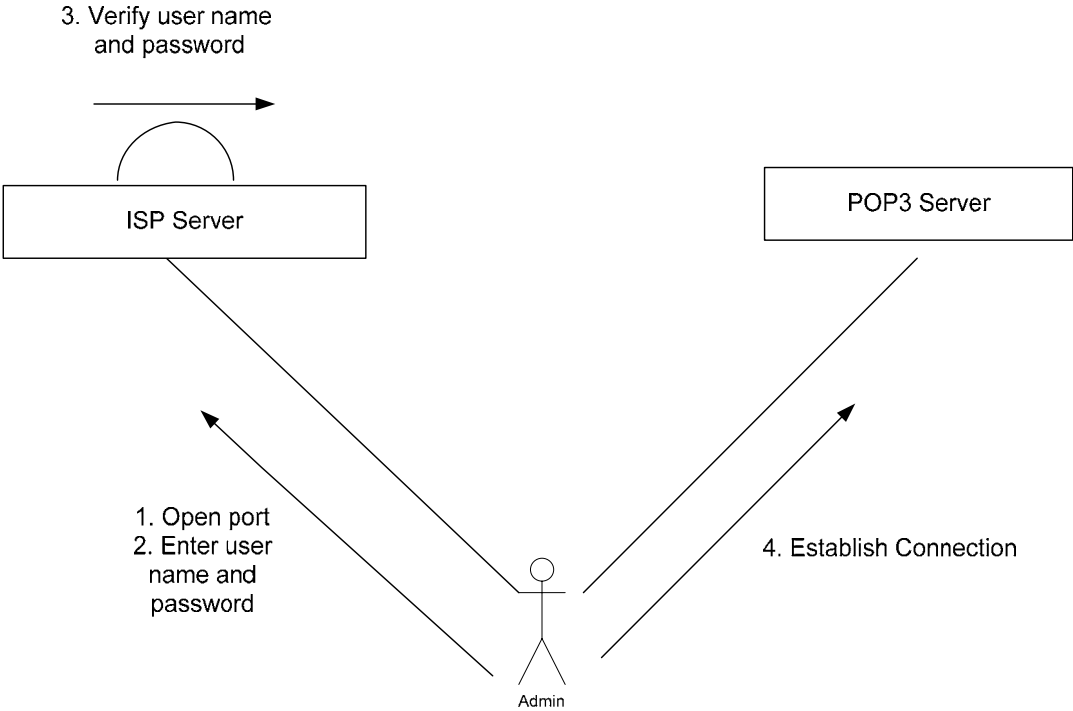
Register Customer



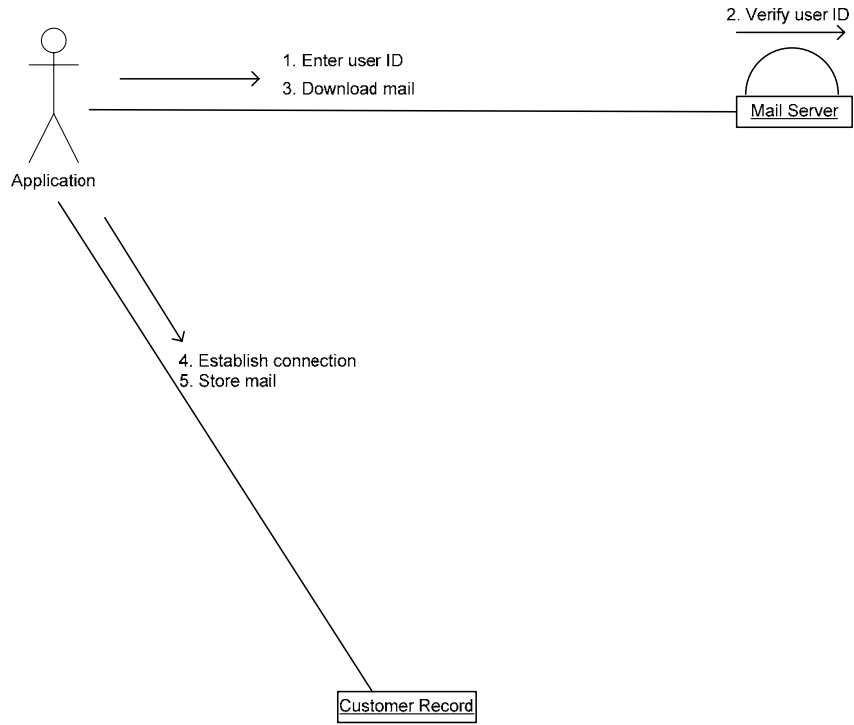
4. Collaboration diagrams



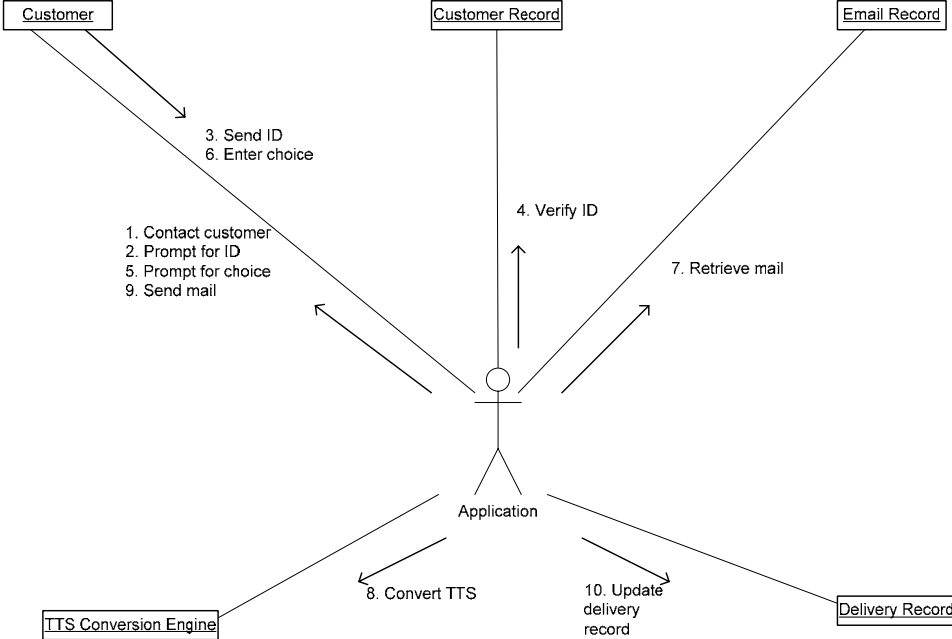
Contact Mail Server



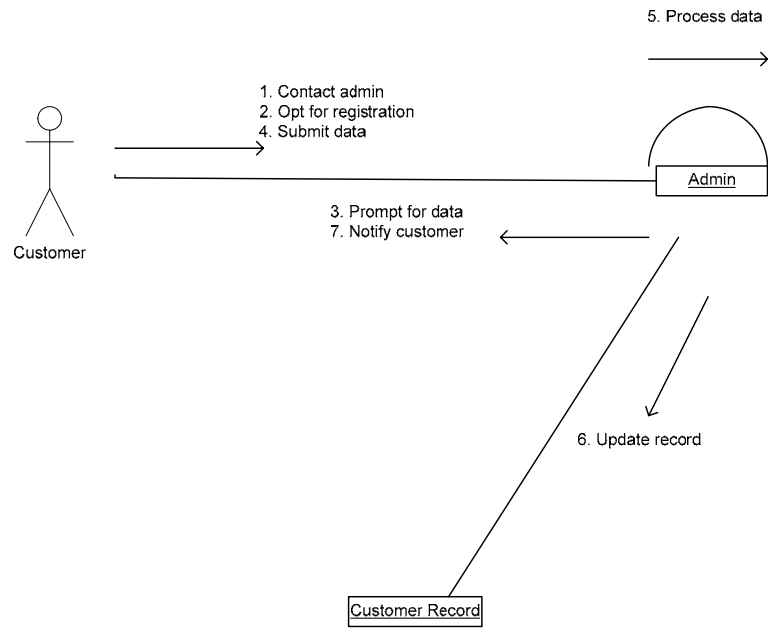
Retrieve & Store Mail



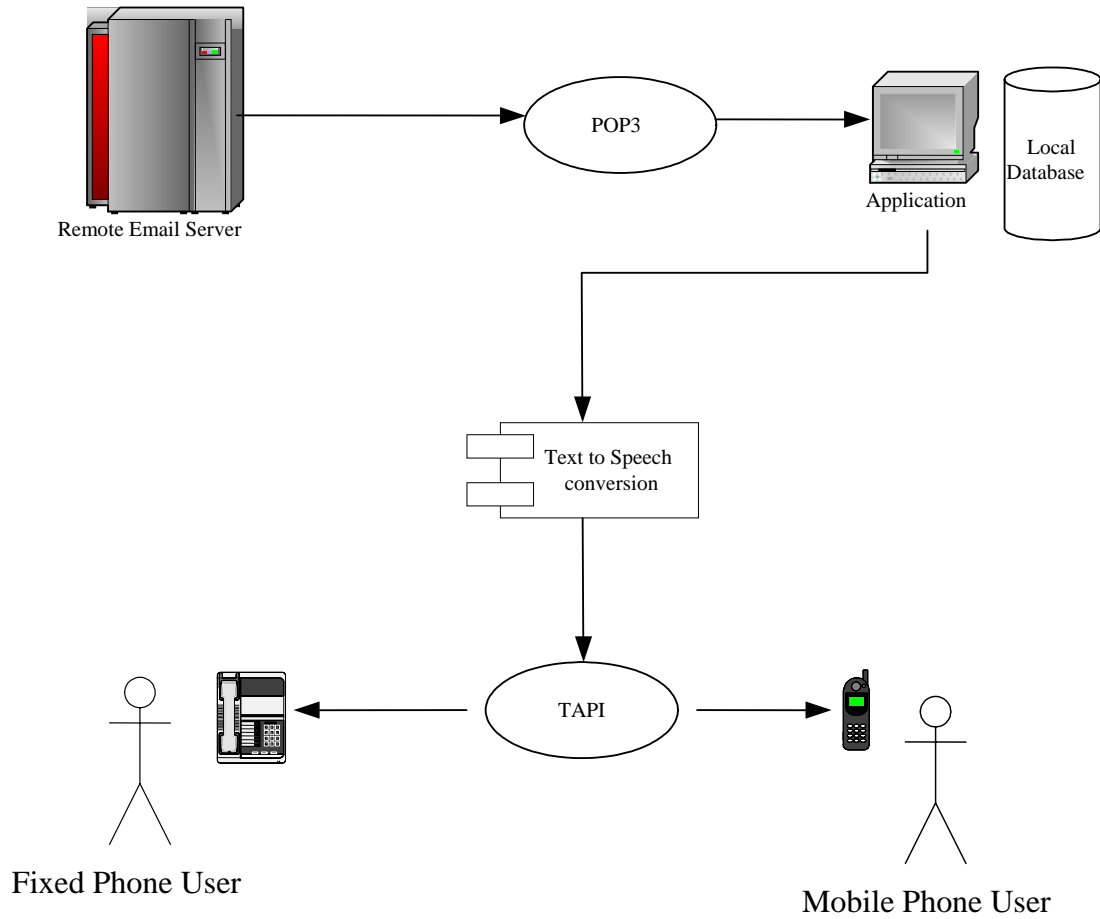
Dispatch Mail

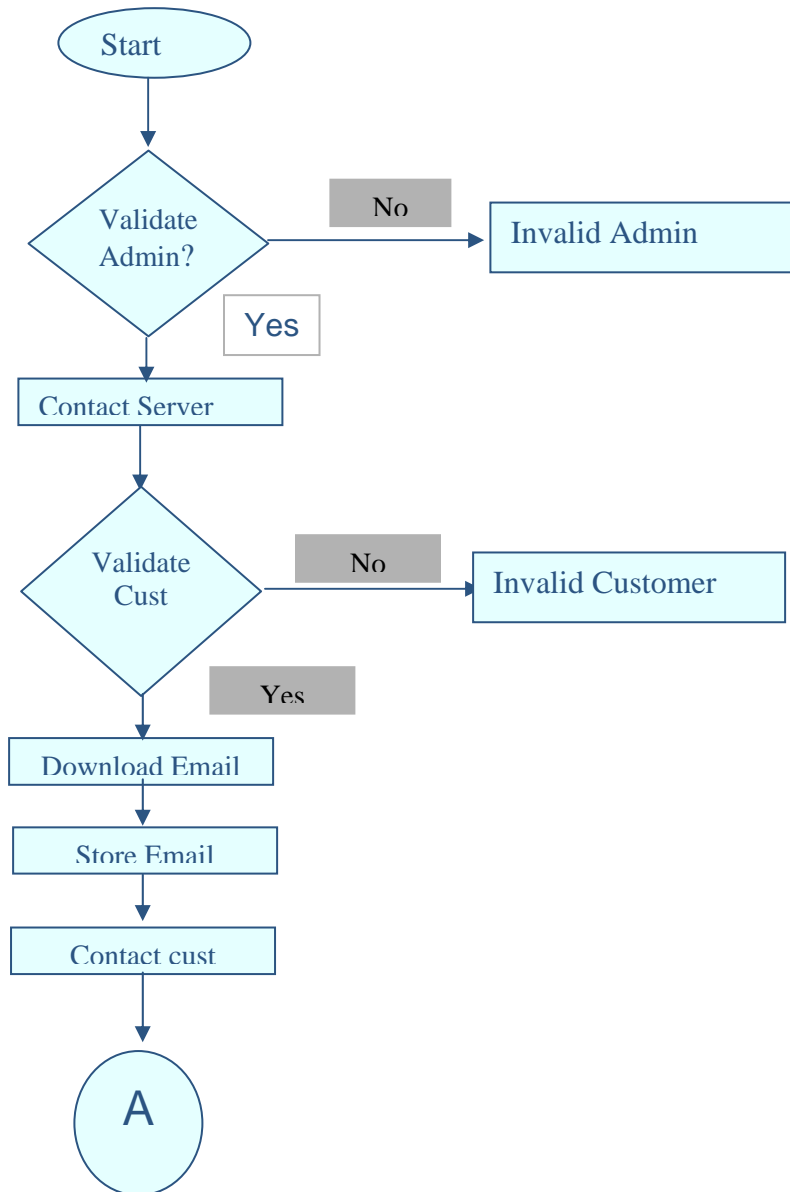


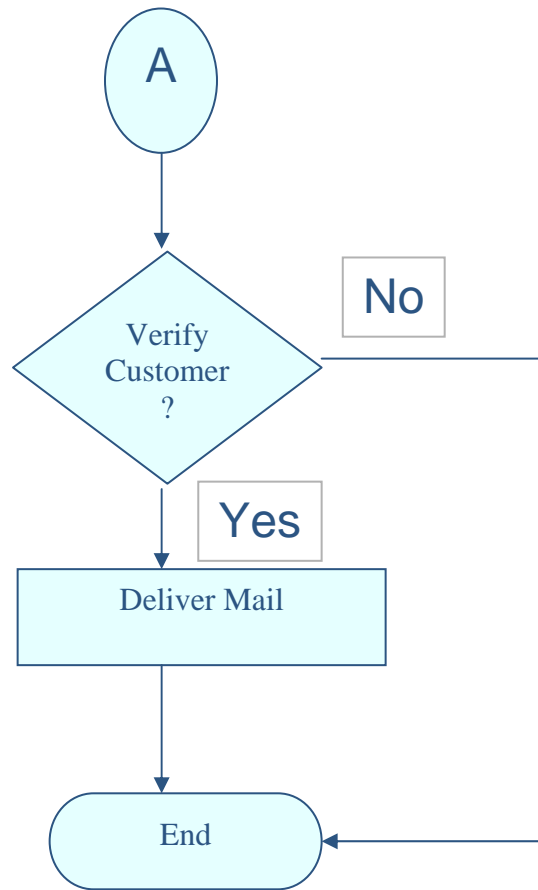
Register Customer



5. Architecture design



6. Flow chart



7. Testing

Software testing is the process of testing the functionality and correctness of software by running it. Software testing is usually performed for one of the following two reasons:

- Defect detection
- Reliability estimation.

The problem of applying software testing to defect detection is that the software can only suggest the presence of flaws, not their absence (unless the testing is exhaustive). The problem of applying software testing to reliability estimation is that the input distribution used for selecting test cases may be flawed. In both of these cases, the mechanism used to determine whether program output is correct (known as an oracle) is often impossible to develop. Obviously the benefit of the entire software testing process is highly dependent on many different pieces. If any of these parts is faulty, the entire process is compromised.

8. Software testing methods (procedural)

There are many ways to conduct software testing, but the most common methods are:-

8.1 Test case design

Test cases should be designed in such a way as to uncover quickly and easily as many errors as possible. They should "exercise" the program by using and producing inputs and outputs that are both correct and incorrect.

Variables should be tested using all possible values (for small ranges) or typical and out-of-bound values (for larger ranges). They should also be tested using valid and invalid types and conditions. Arithmetical and logical comparisons should be examined as well, again using both correct and incorrect parameters.

The objective is to test all modules and then the whole system as completely as possible using a reasonably wide range of conditions.

8.2 White-box testing

White box method relies on intimate knowledge of the code and a procedural design to derive the test cases. It is most widely utilized in unit testing to determine all possible paths within a module, to execute all loops and to test all logical expressions.

Using white-box testing, the software engineer can

- Guarantee that all independent paths within a module have been exercised at least once.
- Examine all logical decisions on their true and false sides.
- Execute all loops and test their operation at their limits.
- Exercise internal data structures to assure their validity.

This form of testing concentrates on the procedural detail. However, there is no automated tool or testing system for this testing method. Therefore even for relatively small systems, exhaustive white-box testing is impossible because of all the possible path permutations.

8.3 Basis path testing

Basis path testing is a white-box technique. It allows the design and definition of a basis set of execution paths. The test cases created from the basis set allow the program to be executed in such a way as to examine each possible path through the program by executing each statement at least once.

To be able to determine the different program paths, the engineer needs a representation of the logical flow of control. The control structure can be illustrated by a flow graph. A flow graph can be used to represent any procedural sign.

The following condition could be represented by the graph on the right:

```
IF a OR b  
THEN proc y  
ELSE proc x  
END IF
```

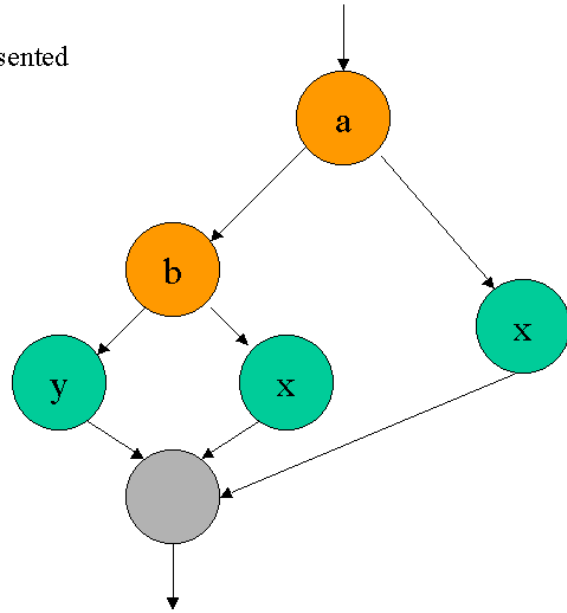


Figure1: Flow graph of an 'If-then-else' statement

Next a metric can be used to determine the number of independent paths. It is called cyclomatic complexity and it provides the number of test cases that have to be designed. This ensures coverage of all program statements.

8.4 Control structure testing

Because basis path testing alone is insufficient, other techniques should be utilized.

- Condition testing can be utilized to design test cases which examine the logical conditions in a program. It focuses on all conditions in the program and includes testing of both relational expressions and arithmetic expressions.
- This can be accomplished using branch testing and/or domain testing methods. Branch testing executes both true and false branches of a condition.

- Domain testing utilizes values on the left-hand side of the relation by making them greater than, equal to and less than the right-hand side value. This method tests both values and the relation operators in the expression.
- Data flow testing method is effective for error protection because it is based on the relationship between statements in the program according to the definition and uses of variables.
- Loop testing method concentrates on validity of the loop structures.

8.5 Black-box testing

Black box on the other hand focuses on the overall functionality of the software. That is why it is the chosen method for designing test cases used in functional testing. This method allows the functional testing to uncover faults like incorrect or missing functions, errors in any of the interfaces, errors in data structures or databases and errors related to performance and program initialization or termination.

9. Testing principles

Following principles were kept in mind while testing our software:-

- Testing should be based on user requirements.
- Testing time and resources are limited. Avoid redundant tests.
- It is impossible to test everything.
- Use effective resources to test. This represents use of the most suitable tools, procedures and individuals to conduct the tests.
- Test planning should be done early.
- Testing should begin at the module

10. Modular testing

The succeeding paragraphs highlight problems encountered during different stages of software development:-

10.1 Login. This feature has been developed for the administrator of the system with following in mind:-

- To provide inbuilt security to the system.
- To validate the administrator.
- To act as an entry point.
- To set a pattern to run and execute the system.

10.1.1 Functioning

This window prompts the administrator of the system to enter Name and Password. Only an authorized administrator having name and password in the database is allowed to run the system.

Initially it was thought that after entering name and password, if matched with the database entries, the administrator would be allowed to run the system. The code is as under:-

```
LsSql = "Select AdminName, AdminPassword from AdminLogin WHERE AdminName
= '' & txtName.Text & '' and AdminPassword = '' & txtPassword.Text & ''"
DataConnection()
If oDr.Read Then
Dim main As New frmwelcome()
main.Show()
Me.Hide()
If end
```

10.1.2 Bugs encountered

- Entering incorrect Name or Password by the administrator.
- Administrator attempts to login without entering name or password.
- Calling of the new window.
- Providing functionality for both mouse click and enter button.
- Prompting messages by the application.

10.1.3 Rectification

- Message box was introduced to prompt messages to the administrator.

- Code was amended as under:-

If e.KeyCode = 13 Then

lsSql = "Select AdminName, AdminPassword from AdminLogin

WHERE AdminName ='" & txtName.Text & "' and AdminPassword

= '" & txtPassword.Text & """

dataConnection()

If oDr.Read Then

Dim main As New frmwelcome()

main.Show()

Me.Hide()

Else

MsgBox("Incorrect name or password.Please try again",

MsgBoxStyle.Exclamation)

End If

End If

10.2 Main screen / Window

This window is the main screen for the administrator. It contains most of the features offered by the application. Different windows and functions can be called while working in this window.

10.2.1 Functioning

Different windows / functions can be called using a variety of functional combinations as follows:-

- Using Dropdown Menu.
- Using Buttons with function icons.
- Using Right Click of the mouse.

Besides calling windows / functions the administrator can also perform a variety of tasks as follows:-

- Can edit text using text editor.

- Can take out print.
- Can open other files / programs.
- Can save files.

10.2.2 Bugs encountered

- Simultaneous calling and hiding of windows.
- Providing multiple functional methods to perform the same task for the reason of backup and ease of use.

10.2.3 Rectification

Code was amended as under:-

(Calling multiple windows using menu item)

```
Private Sub MenuItem11_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem11.Click
```

```
Dim add As AddAdmin = New AddAdmin()
```

```
add.cbDelete.Visible = False
```

```
add.cbAdd.Location = New System.Drawing.Point(40, 8)
```

```
add.cbSave.Location = New System.Drawing.Point(112, 8)
```

```
add.cbClose.Location = New System.Drawing.Point(184, 8)
```

```
add.cbSave.Text = "Insert"
```

```
add.Show()
```

```
End Sub
```

```
Private Sub MenuItem18_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem18.Click
```

```
Dim retrieve As New frmMailRetrieval()
```

```
retrieve.Show()
```

End Sub

Private Sub MenuItem12_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MenuItem12.Click

Dim add As AddAdmin = New AddAdmin()

add.cbDelete.Visible = False

add.cbSave.Text = "update"

add.cbAdd.Location = New System.Drawing.Point(40, 8)

add.cbSave.Location = New System.Drawing.Point(112, 8)

add.cbClose.Location = New System.Drawing.Point(184, 8)

add.Show()

End Sub

Private Sub MenuItem13_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MenuItem13.Click

Dim add As AddAdmin = New AddAdmin()

add.cbSave.Visible = False

add.cbAdd.Visible = False

add.cbDelete.Location = New System.Drawing.Point(80, 8)

add.cbClose.Location = New System.Drawing.Point(152, 8)

add.Show()

End Sub

(Calling multiple windows using context menu)

Private Sub tbSystem_ButtonClick(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) Handles tbSystem.ButtonClick

Select Case tbSystem.Buttons.IndexOf(e.Button)

Case 0


```
Dim tcbAdmin As New AddAdmin()  
tcbAdmin.Show()  
' Insert code to open the file.
```

Case 1

```
Dim tcbRetrieveMail As New frmMailRetrieval()  
tcbRetrieveMail.Show()
```

Case 3

```
Dim tcbTapi As New frmTapi()  
tcbTapi.Show()
```

Case 4

```
Dim tcbSapi As New frmBasiccts()  
tcbSapi.Show()
```

Case 7

```
Dim tcbLogin As New frmLogin()  
tcbLogin.Show()
```

Case 6

```
Dim tcbCustomer As New frmCustomerProfile()  
tcbCustomer.Show()
```

End Select

End Sub

10.3 Administrator record

This window is basically meant for the following purpose:-

- To view the record of the administrators.
- To save the new record.
- To update the record.
- To delete the record.

10.3.1 Functioning

This window allows the administrator to view, save, update or delete the administrator record in the database. It also provides instant record viewing as soon as it is inserted, updated, saved or deleted.

10.3.2 Bugs encountered

- Displaying the updated record as soon as it is entered.
- Leaving a field vacant.
- Prompting messages by the system.
- Position of cursor after entering / updating or deleting a record.
- Enabling / disabling of appropriate fields.
- Entering a record with incorrect name or password.
- What to be done if incorrect password is entered in the Confirm Password field?

10.3.3 Rectification

- Message box is used to prompt the system messages.
- Different checks are used to cater for the various contingencies.
- ADO.NET features are used to dynamically access the database.
- Some important code highlights are as under:

(To cater for blank Name or Password)

```
SQLConn = New System.Data.SqlClient.SqlConnection(
    connectionString()
SQLConn.Open()
If cbAdd.Enabled = True Then
    lssSql = "update AdminLogin set AdminName='" & txtName.Text & "' ,
AdminPassword='" & txtPassword.Text & "' where AdminId='" & txtID.Text & "'"
    oCmd.Connection = SQLConn
```

oCmd.CommandText = lssSql

If txtName.Text = "" Or txtPassword.Text = "" Then

MsgBox("User Name or Password Can't be blank", MsgBoxStyle.Exclamation)

(To cater for confirm Password field)

Else

If txtPassword.Text = txtConfirmpassword.Text Then

oDr = oCmd.ExecuteReader

MsgBox("Record Updated", MsgBoxStyle.Exclamation)

If txtName.Text = lstAdmin.SelectedItem Then

txtID.Text = ""

txtName.Text = ""

txtPassword.Text = ""

txtConfirmpassword.Text = ""

txtName.Focus()

Else

lstAdmin.Items.Remove(lstAdmin.SelectedItem)

lstAdmin.ClearSelected()

lstAdmin.Items.Add(txtName.Text)

txtID.Text = ""

txtName.Text = ""

txtPassword.Text = ""

txtConfirmpassword.Text = ""

txtName.Focus()

End If

Else

MsgBox("Please enter the password again", MsgBoxStyle.Exclamation)

(To cater for empty field)

If txtName.Text = "" Or txtPassword.Text = "" Then

*MsgBox("Please enter user name and password. These fields can't be empty ",
MsgBoxStyle.Critical)*

(To cater for both mouse click and Enter Button)

If e.KeyCode = 13 Then

If cbAdd.Enabled = False Then

*lsSql = "Insert into AdminLogin(AdminName,AdminPassword) Values('" &
txtName.Text & "', '" & txtPassword.Text & "')"*

oCmd.Connection = SQLConn

oCmd.CommandText = lsSql

If txtPassword.Text = txtConfirmpassword.Text Then

oDr = oCmd.ExecuteReader

MsgBox("One record inserted", MsgBoxStyle.Exclamation)

txtConfirmpassword.Text = ""

txtPassword.Focus()

End If

lstAdmin.Enabled = True

cbDelete.Enabled = True

```

        cbClose.Enabled = True
    End If
End If
SQLConn.Close()

```

10.4 Customer record / Customer profile

These features/windows are designed to provide customer record for the consumption of the administrator so that he is able to perform basic functions related to customer record/profile. These two separate windows are used in conjunction with each other to make a meaningful executing combination.

10.4.1 Functioning

These features/windows are equipped with following facilities:-

- Insert new customer record.
- Update customer record.
- View customer record.
- Delete customer record.
- List View control functions.
- Double click function

10.4.2 Bugs encountered

- Insertion, updation, deletion in a single session.
- Calling a window within another window, both using different controls (List View Control and Text Box Control).
- Implementation of List View Operations.

10.4.3 Rectification

The workable solution to all above mentioned bugging problems was enthusiastically pursued. It involved using some new concepts of calling a window within another window and executing ListView functions. Solution came in the form of using the two above mentioned windows / features simultaneously.

(To initialize object “locustomerprofile” in customer Record Window)

```
Public Sub New()  
    MyBase.New()  
    InitializeComponent()  
    End Sub  
Public Sub New(ByVal loCustomerProfile)  
    Me.loCustomerProfile = loCustomerProfile  
    InitializeComponent()  
    End Sub  
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)  
If disposing Then  
    If Not (components Is Nothing) Then  
        components.Dispose()  
    End If  
End If  
    MyBase.Dispose(disposing)  
End Sub
```

(To initialize object “pocustomerprofile” in customer profile Window)

```
Public Sub New()  
    MyBase.New()  
  
    InitializeComponent()  
  
End Sub
```

```

Public Sub New(ByVal poCustomer)
    Me.poCustomer = poCustomer
    InitializeComponent()

End Sub

```

(To ensure password field and confirm password field are the same)

```

If txtPassword.Text = txtConfirmPassword.Text Then
    dataConnection()
    MsgBox("one record inserted")

```

Else

```

    MsgBox("Please re-enter the password again")

```

(To ensure that one window is hidden when the other one is displayed)

```

Dim populateCustomer As New frmCustomerProfile(Me)

```

```

    populateCustomer.Show()
    loCustomerProfile.Hide()

```

(To cater for Enter Button to work)

```

If e.KeyCode = 13 Then

```

```

    If txtID.Text = "" Then
        If txtName.Text = "" Or txtPassword.Text = "" Or txtConfirmPassword.Text =
"" Or txtEmail.Text = "" Or txtTelephone.Text = "" Or txtAddress.Text = "" Then
            MsgBox("No Field can be Blank", MsgBoxStyle.Exclamation)

```

Else

```

        lssSql = "Insert into CustomerLogin(CustomerName,
CustomerPassword, CustomerEmail, CustomerTelephonenumber,
CustomerAddress) Values('" & txtName.Text & "', '" & txtPassword.Text & "', '"
& txtEmail.Text & "', '" & txtTelephone.Text & "', '" & txtAddress.Text & "')"

```

(To display customer record in list view)

Do While oDr.Read

```

lvwCustomer.Items.Add(oDr.Item("CustomerID"))
With lvwCustomer.Items(lnIndex)
    .SubItems().Add(oDr.Item("CustomerName"))
    .SubItems().Add(oDr.Item("CustomerPassword"))
    .SubItems().Add(oDr.Item("CustomerEmail"))
    .SubItems().Add(oDr.Item("CustomerTelephonenumber"))
    .SubItems().Add(oDr.Item("CustomerAddress"))

```

End With

```

    lnIndex = lnIndex + 1

```

Loop

(To call customer Record Window in customer profile Window)

Dim editCustomer As New frmCustomer(Me)

```

    txtID.Text =

```

```

lvwCustomer.Items(lvwCustomer.SelectedIndices.Item(0).ToString).Text

```

```

    gnCustomerID = Val(txtID.Text)

```

```

    editCustomer.Show()

```

If response = MsgBoxResult.Yes Then


```
'      populate()  
End If
```

10.5 Mail retrieval

This feature/window is one of the most important aspects of this application. It takes user email address and password, connects to the POP3 compliant server and then downloads the mail.

10.5.1 Functioning

This window connects to the POP3 compliant server using socket connection. Once connected it retrieves mail as specified in the address.

10.5.2 Bugs encountered

- Size of the received messages had to be specified (Lengthy messages through voice mail were inappropriate).
- Direct method of connecting to POP3 server had to be found.

10.5.3 Rectification

- Use of sockets.
- Setting limit for the messages to be downloaded
- Appropriate changes in the code

10.6 Text to speech conversion

This is another very important feature of the application which converts text messages to voice for onward transmission to fixed or mobile phone .

10.6.1 Functioning

This feature basically plays the role of a conversion engine of the application. It utilizes the built in functionality of the windows operating system.

10.6.2 Bugs encountered

- Pauses in conversation at appropriate places.
- Speaking speed.
- Speaking of some unwanted characters during speech.

10.6.3 Rectification

(Removing unnecessary characters from the speech)

```
Dim myString As String = txtFrom.Text
```

```
txtFrom.Text = Replace(myString, "<", " ")
```

```
Dim myString1 As String = txtFrom.Text
```

```
txtFrom.Text = Replace(myString1, ">", " ")
```

(Incorporating necessary pauses in the speech)

```
TextToSpeech1.Speak("from")
```

```
TextToSpeech1.Speak(txtFrom.Text)
```

```
TextToSpeech1.Speak("To")
```

```
TextToSpeech1.Speak(txtTo.Text)
```

```
TextToSpeech1.Speak("Subject")
```

```
TextToSpeech1.Speak(txtSubject.Text)
```

```
TextToSpeech1.Speak("Contents")
```

```
TextToSpeech1.Speak(txtContents.Text)
```

Chapter 9

REMOTE ACCESS

1. Introduction

As has been discussed in the preceding chapters that the system is not fully automated and because of this, manual interaction of the administrator with the system is quite pronounced. This interaction from registering the customer for the first time to establishing the contact with the ISP and the POP3 server and dispatching the mail to the customer. The first one in the above list i.e registering the customer for the first time obviously raises privacy concerns of the customer, as apart from other profile information he will have to furnish information about his email address and password as well. To address this genuine concern of the customer it was decided to provide him with remote access to the system where from he can change his password through a registration form which once submitted will update his record, bypassing the administrator. This forms the basis of our decision to develop and launch a website as a complementary module to our application.

2. Links available

The website offers following links:-

- **Home**. Displays the title of the system together with the other links available.
- **About site**. Gives an account of the features available in the website.
- **About application**. Gives an overview of the application which includes:-
 - [Project team](#). Gives a brief account of the development team.
 - [Basic idea](#). Describes the motivation and background for undertaking this very project.
 - [Operation](#). Describes the operational flow of the system.
 - [Technologies](#). Gives an overview of the technologies employed to develop the system.
 - [Salient features](#). Outlines the basic facilities offered to the subscribers of the system.
 - [Limitations](#). Highlights the boundaries and limitations of the system which can be taken care of as part of future enhancements.

- ***Login***. Prompts the customer to enter his ID and password. On successful validation the customer is taken to a form which can be used to update any profile information.
- ***Guest book***. People visiting for the first time can record their comments and suggestions as well as view the comments given by other visitors.
- ***Acknowledgements***. Acknowledge the help and guidance rendered by different people which played an instrumental role in successful completion of the project.

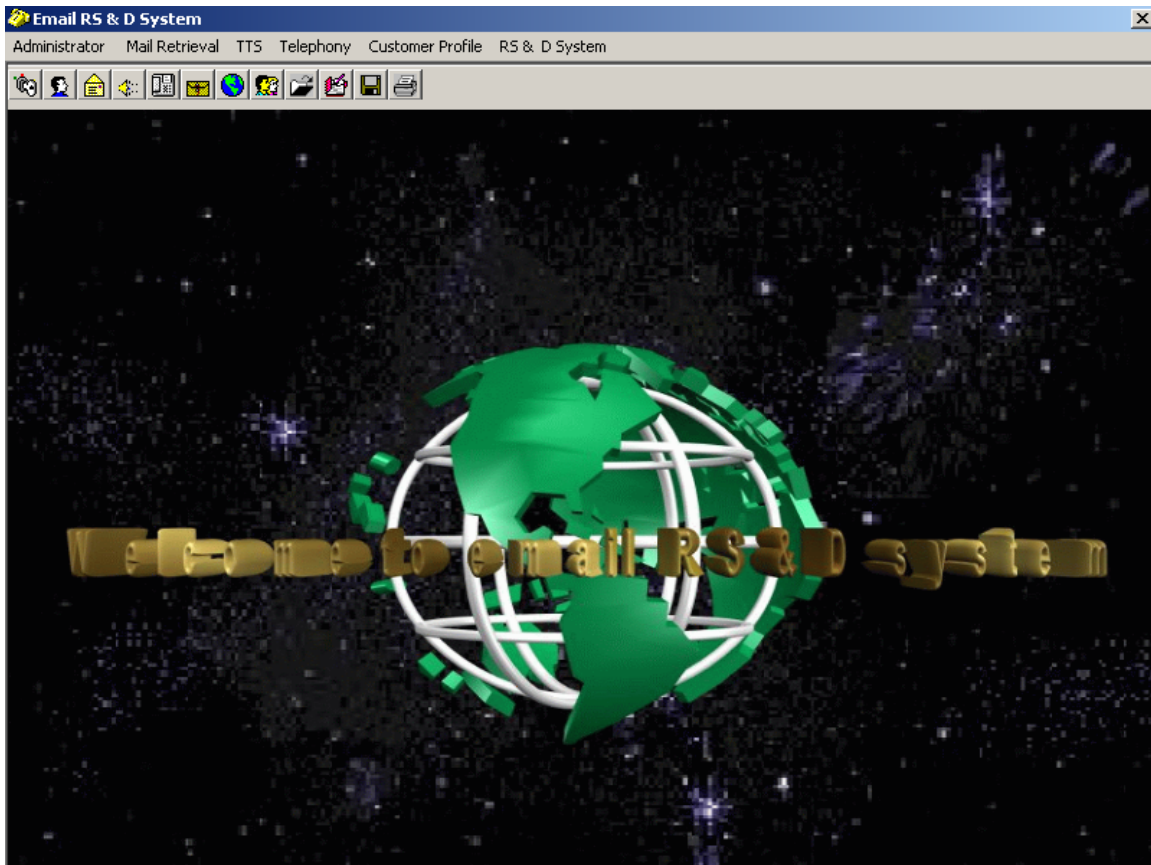
CONCLUSION

As a whole, the project has been a success as it gave us a valuable chance to acquire knowledge in the latest fields of .NET and voice telephony. The rich experience gained during the course of this project will help us in our forthcoming endeavours. This project will indeed be a milestone in our academic and professional careers.

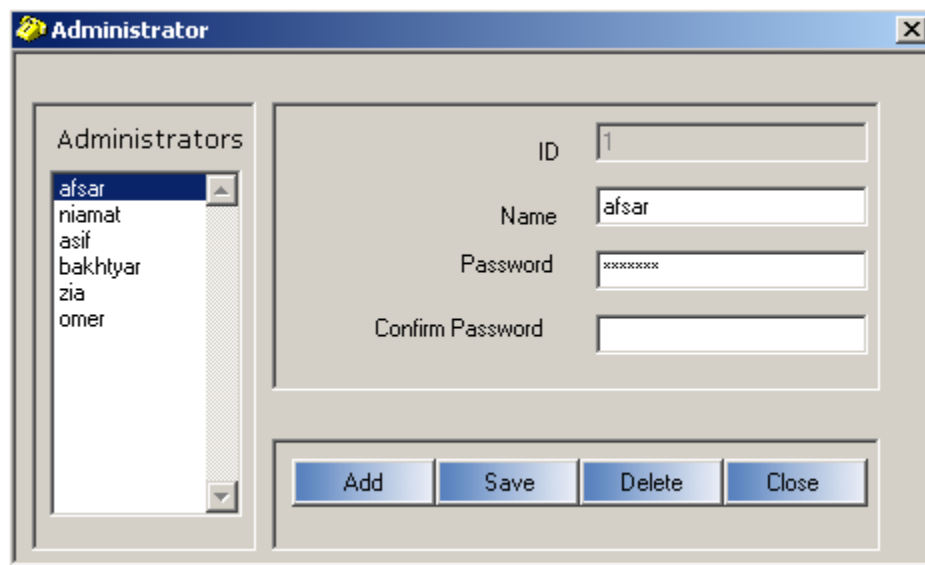
While analyzing designing and implementing this project, we tried to apply all we had studied in different courses throughout this undergraduate degree program. This exercise not only allowed us to implement the theoretical concepts of computer science, but also provided us with an excellent opportunity to revise and refresh everything in great detail. This has made us ready for the challenges of practical life and abreast of the cutting edge technological breakthroughs.

ILLUSTRATIONS

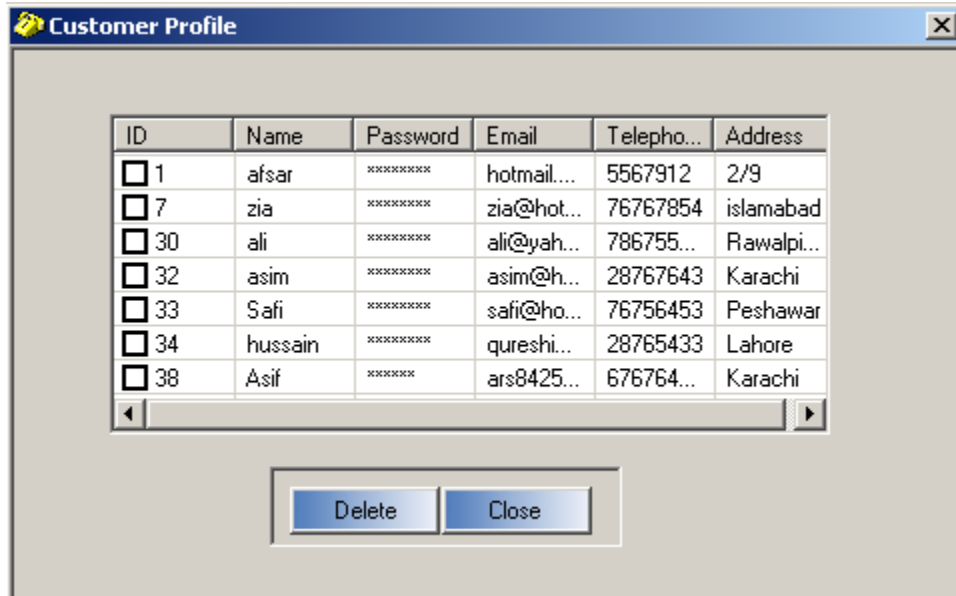
1. Main GUI



2. Admin view



3. Customer profile

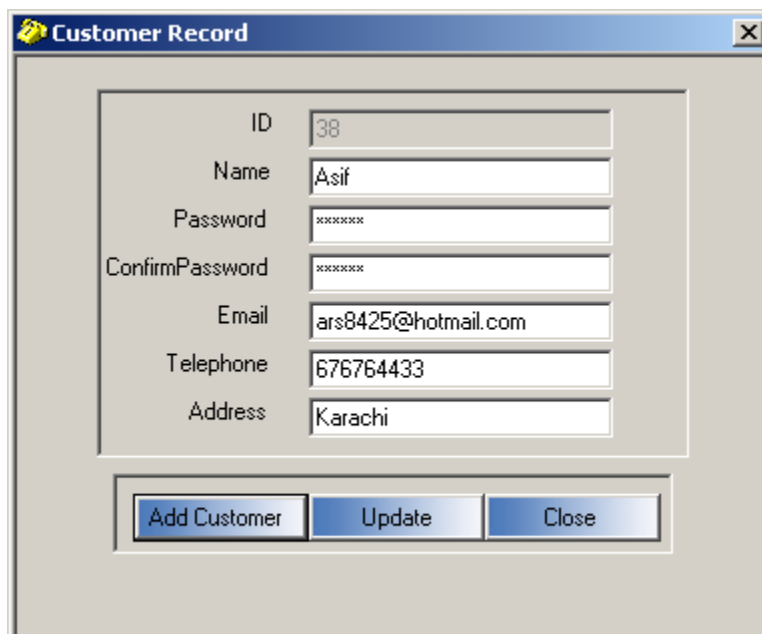


Customer Profile

ID	Name	Password	Email	Telepho...	Address
<input type="checkbox"/> 1	afsar	*****	hotmail...	5567912	2/9
<input type="checkbox"/> 7	zia	*****	zia@hot...	76767854	islamabad
<input type="checkbox"/> 30	ali	*****	ali@yah...	786755...	Rawalpi...
<input type="checkbox"/> 32	asim	*****	asim@h...	28767643	Karachi
<input type="checkbox"/> 33	Safi	*****	safi@ho...	76756453	Peshawar
<input type="checkbox"/> 34	hussain	*****	qureshi...	28765433	Lahore
<input type="checkbox"/> 38	Asif	*****	ars8425...	676764...	Karachi

Delete Close

4. Customer record



Customer Record

ID: 38

Name: Asif

Password: *****

ConfirmPassword: *****

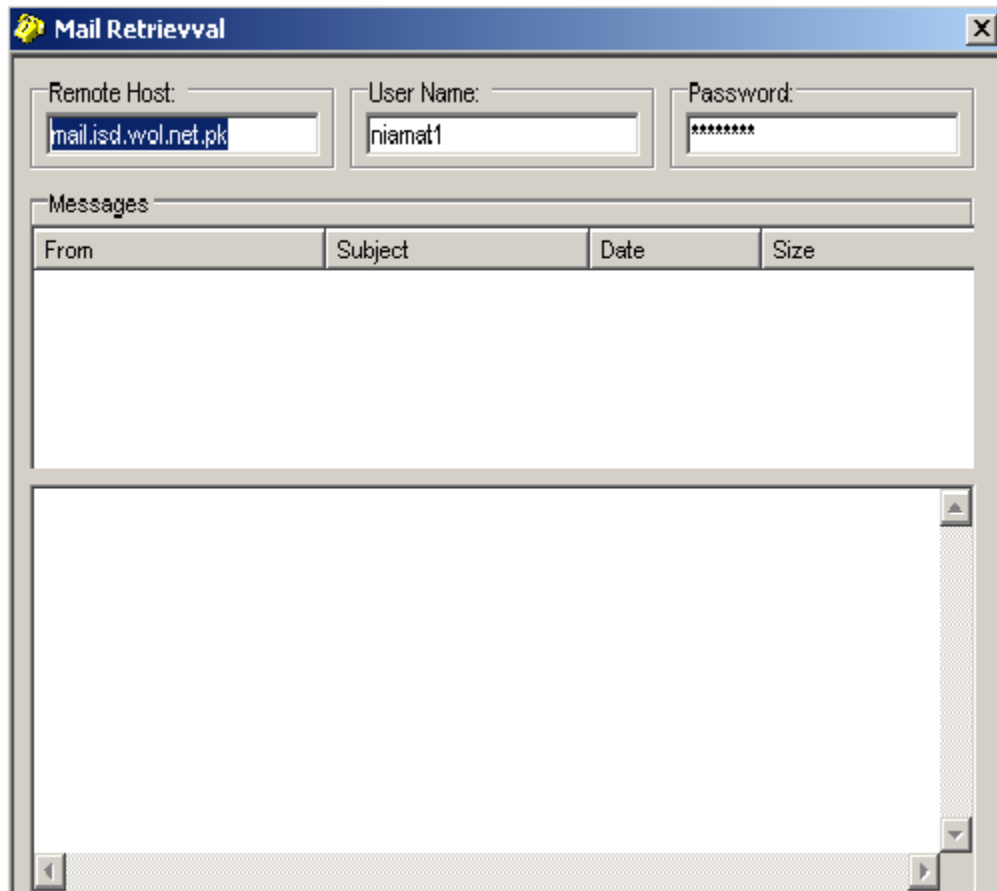
Email: ars8425@hotmail.com

Telephone: 676764433

Address: Karachi

Add Customer Update Close

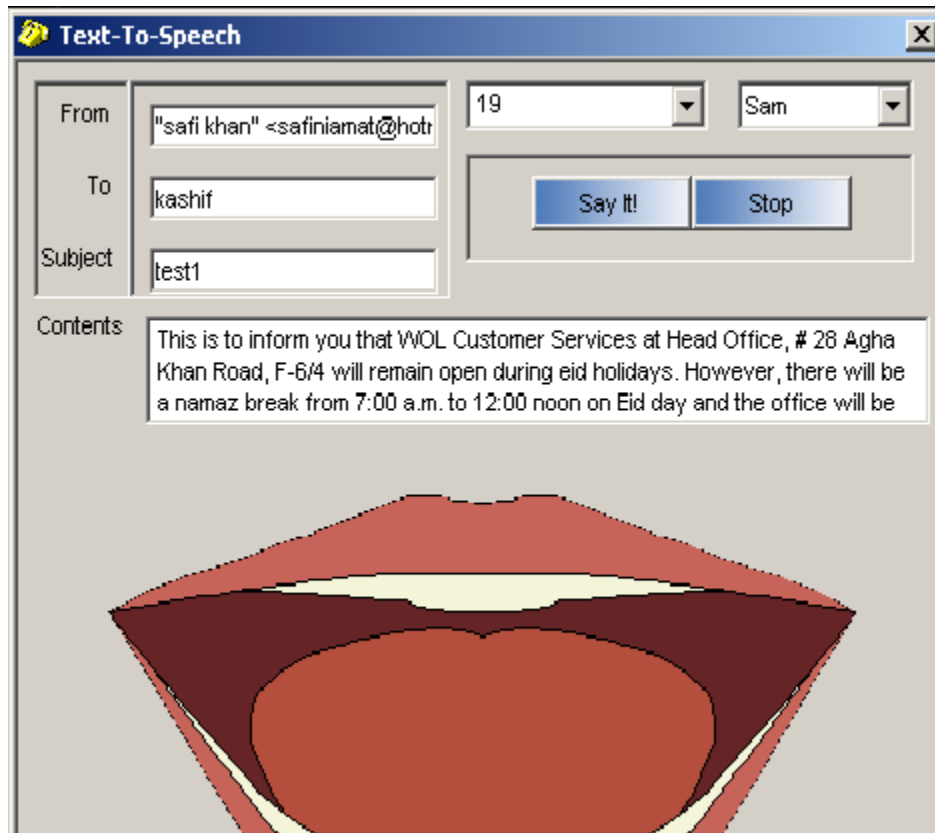
5. Mail retrieval



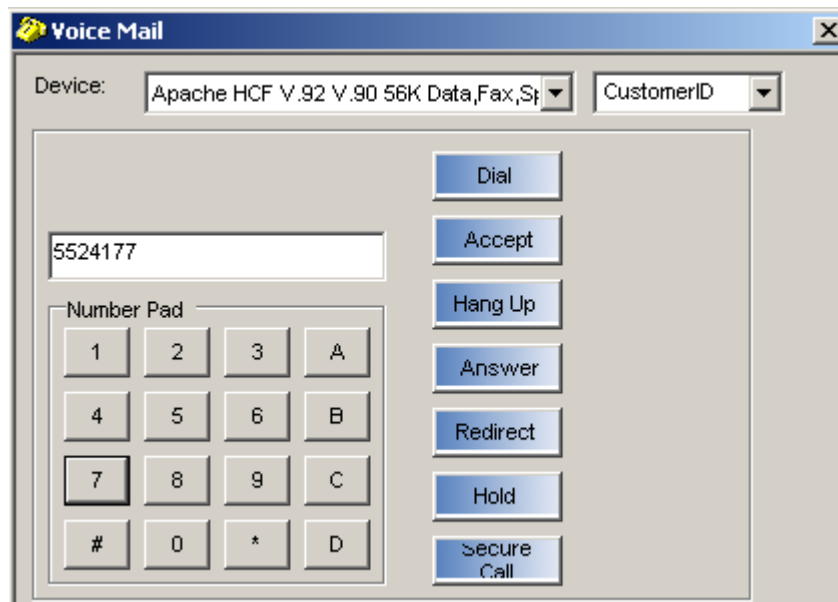
The image shows a 'Mail Retrieval' dialog box with a blue title bar and a close button. It contains three input fields for configuration: 'Remote Host' with the value 'mail.isd.wol.net.pk', 'User Name' with the value 'niamat1', and 'Password' with a masked value of seven asterisks. Below these fields is a 'Messages' section with a table header containing 'From', 'Subject', 'Date', and 'Size'. The table body is empty. At the bottom of the dialog is a large, empty scrollable area with a vertical scrollbar on the right and horizontal scrollbars at the bottom.

From	Subject	Date	Size
------	---------	------	------

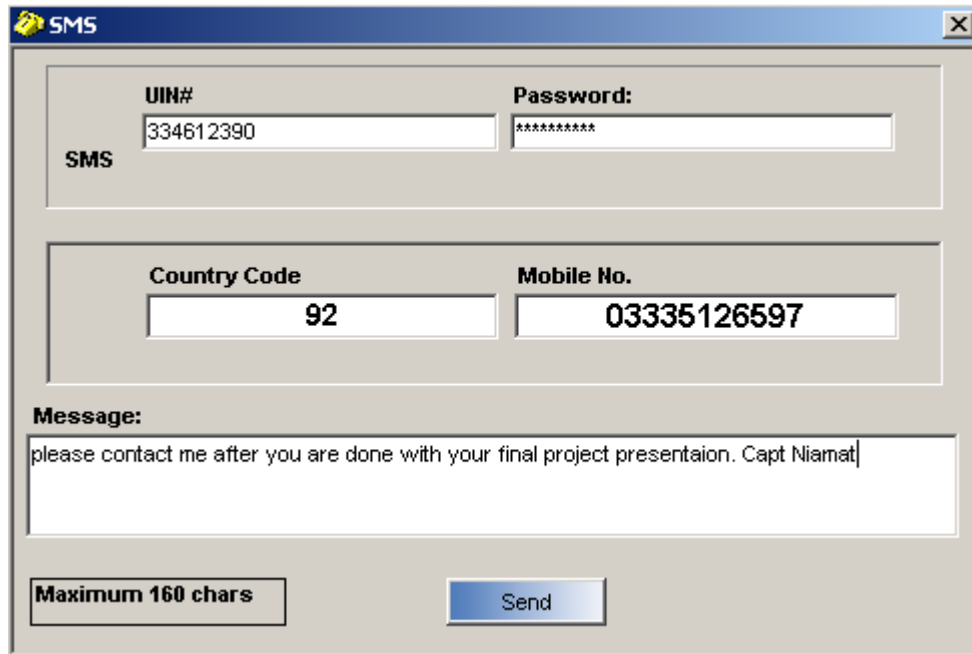
6. TTS



7. Voice mail



8. SMS



The image shows a screenshot of a software application window titled "SMS". The window has a blue title bar with a yellow icon on the left and a close button on the right. The main content area is light gray and contains several input fields and a message box. The "UIN#" field contains "334612390" and the "Password:" field contains "*****". The "Country Code" field contains "92" and the "Mobile No." field contains "03335126597". The "Message:" field contains the text "please contact me after you are done with your final project presentaion. Capt Niamat". At the bottom left, there is a box labeled "Maximum 160 chars". At the bottom right, there is a blue "Send" button.

UIN#	Password:
334612390	*****

SMS

Country Code	Mobile No.
92	03335126597

Message:

please contact me after you are done with your final project presentaion. Capt Niamat

Maximum 160 chars

Send

9. Composing

To kashif

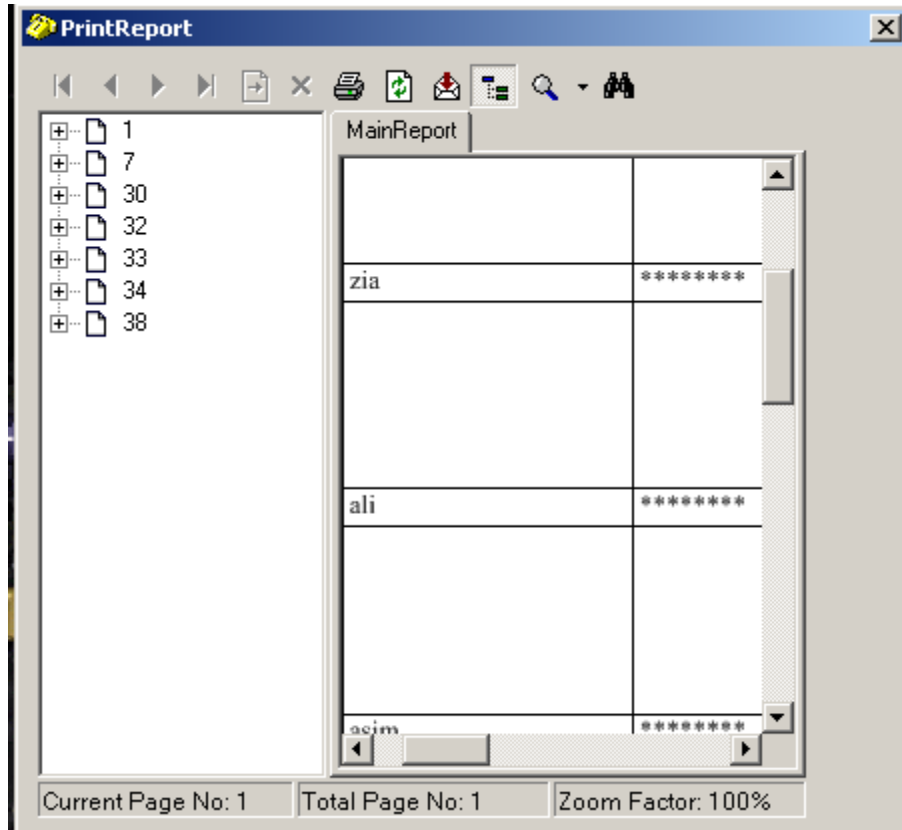
From "safi khan" <safiniamat@hotmail.com>

Subject test1

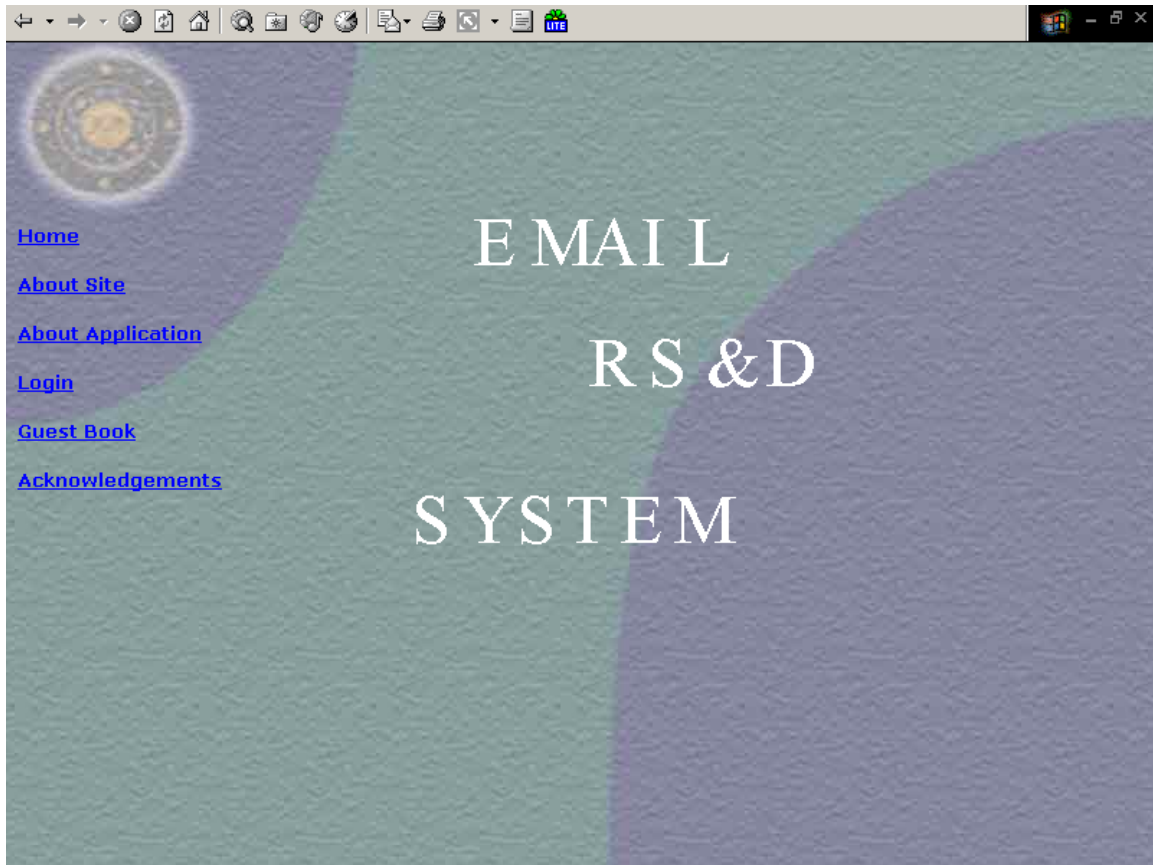
Body 19

Road, F-6/4 will remain open during eid holidays. However, there will be a namaz break from 7:00 a.m. to 12:00 noon on Eid day and the office will be opened afterwards.
Moreover, WOL Net Dokaan will remain closed on Eid days. WOLACCESS, free courier Service by WOL, will not be available from Dec 5 - 7, 2002 and will resume on its normal schedule afterwards. If you have any queries, please feel free to contact us at 111600111. WOL always looks forward to providing you with the best services and maximum convenience.
Regards,

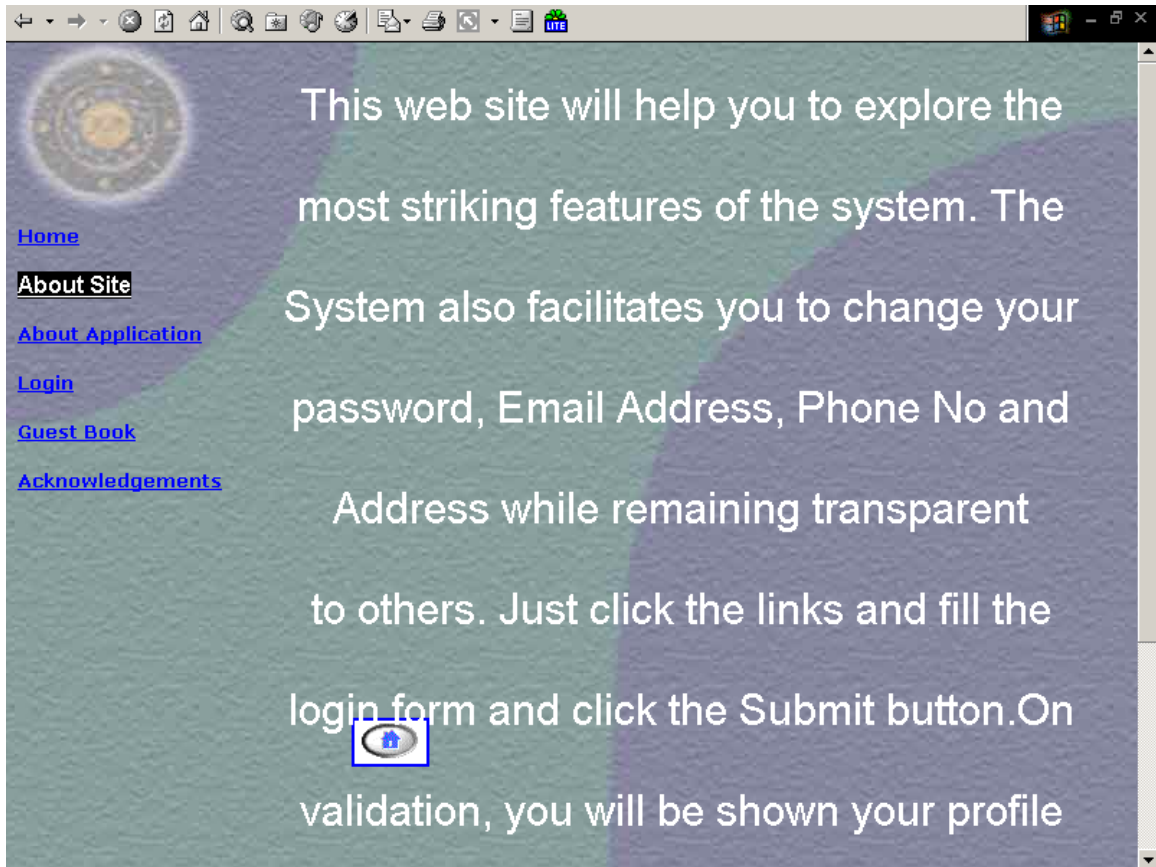
Out Going server address mail.isd.wol.net.pk

11. Print report

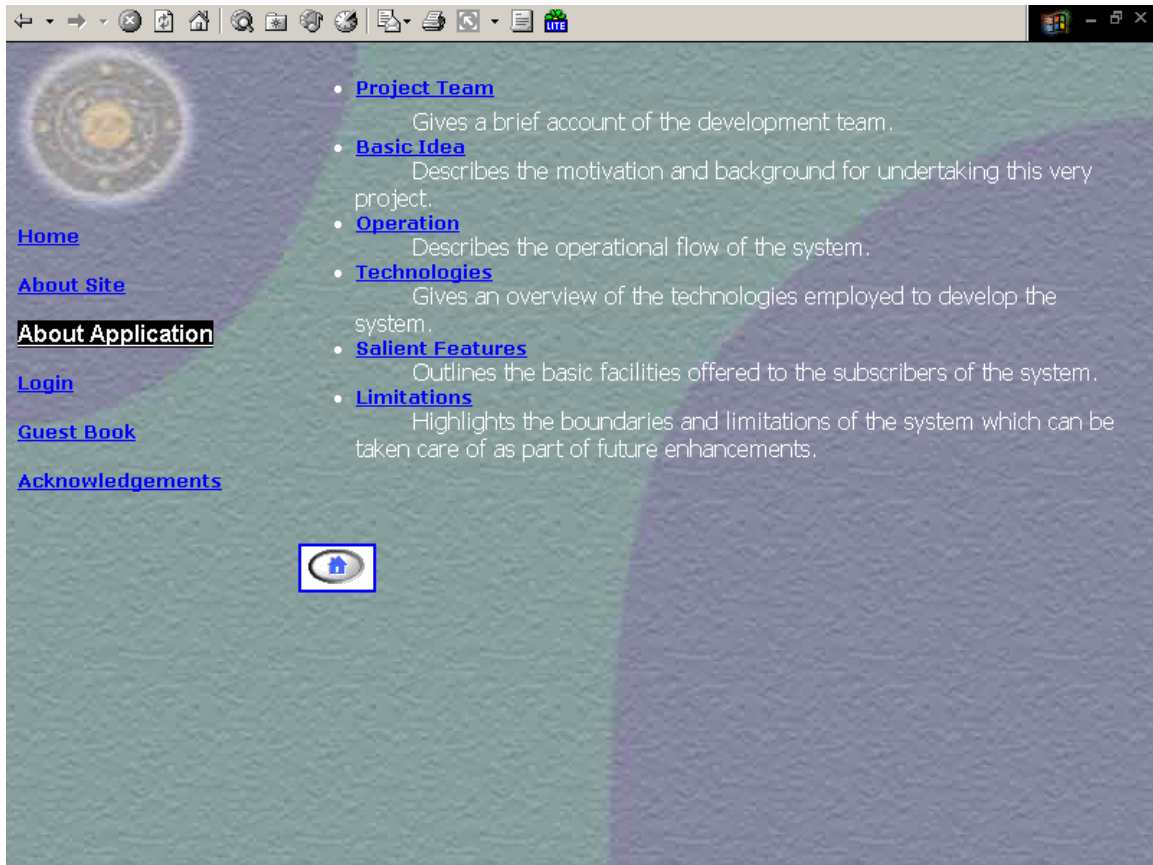
12. Home page



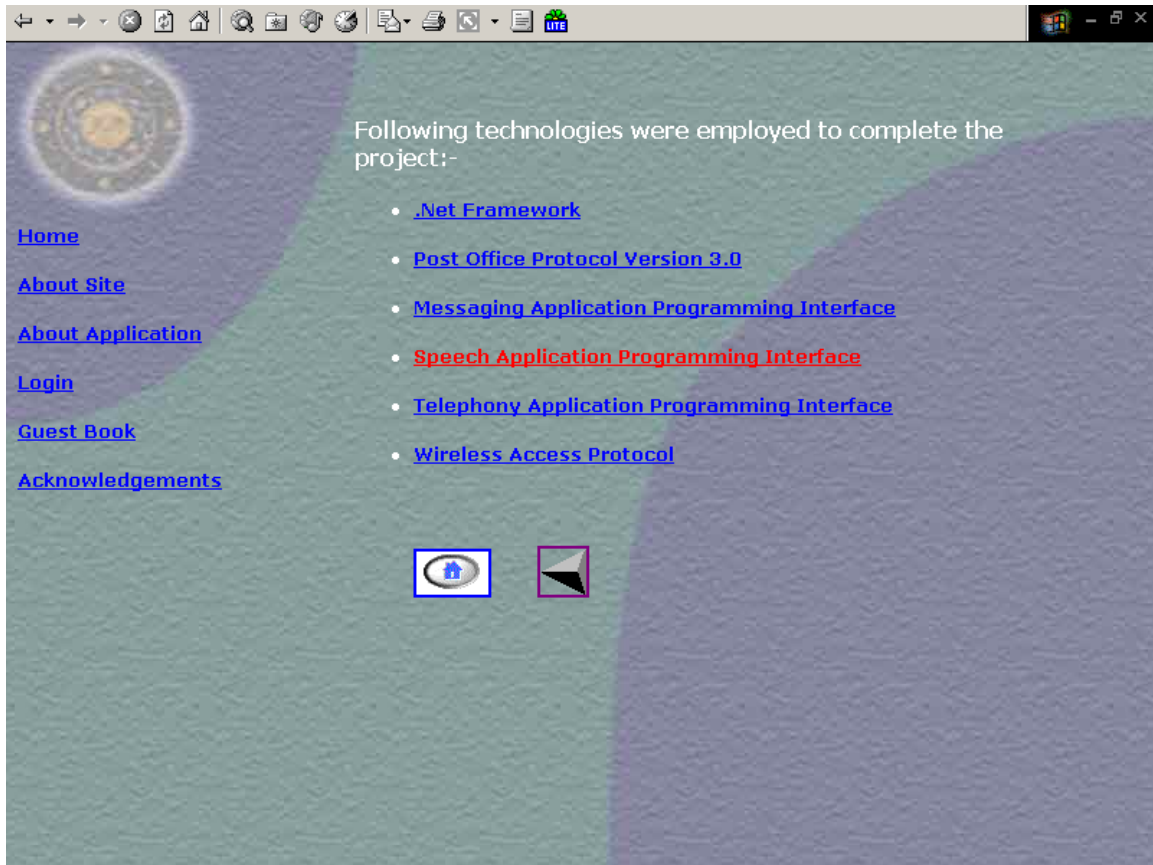
13. About site



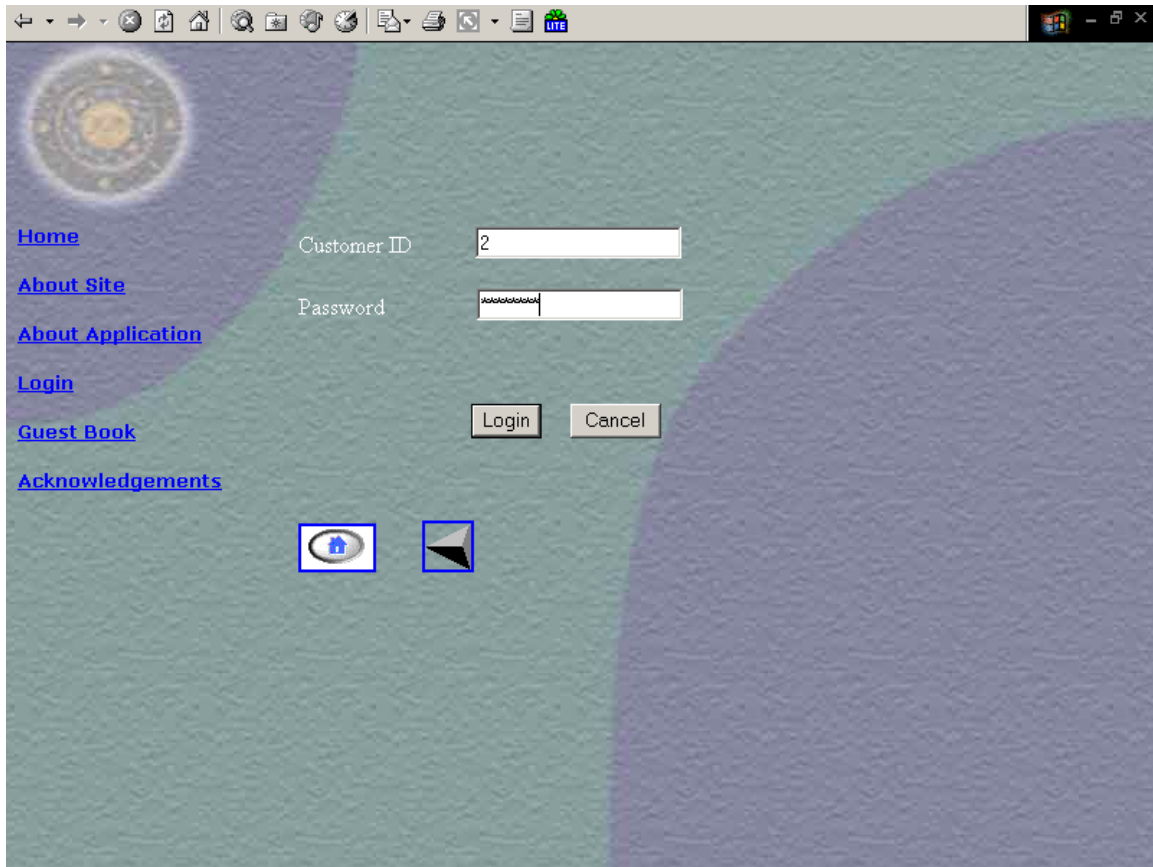
14. About application



15. Technologies used



16. Login



17. Customer profile

The screenshot shows a web browser window with a customer profile form. The browser's address bar is empty, and the Windows taskbar is visible at the top. The form is set against a background with a circular graphic on the left and a green and blue abstract design. The form fields are as follows:

ID	<input type="text" value="1"/>
Name	<input type="text" value="Afsar"/>
Old Password	<input type="password" value="*****"/>
New Password	<input type="password" value="*****"/>
Confirm Password	<input type="password" value="*****"/>
Telephone	<input type="text" value="5567912"/>
Email	<input type="text" value="aqurash@isd.wol.net.pk"/>
Address	<input type="text" value="2/9"/>

Navigation links on the left side:

- [Home](#)
- [About Site](#)
- [About Application](#)
- [Login](#)
- [Guest Book](#)
- [Acknowledgements](#)

Buttons at the bottom of the form:

-
-

At the bottom left, there are two icons: a home icon and a cursor icon.

18. Guest book

We welcome your valuable comments and suggestions.

[Home](#)
[About Site](#)
[About Application](#)
[Login](#)
[Guest Book](#)
[Acknowledgements](#)

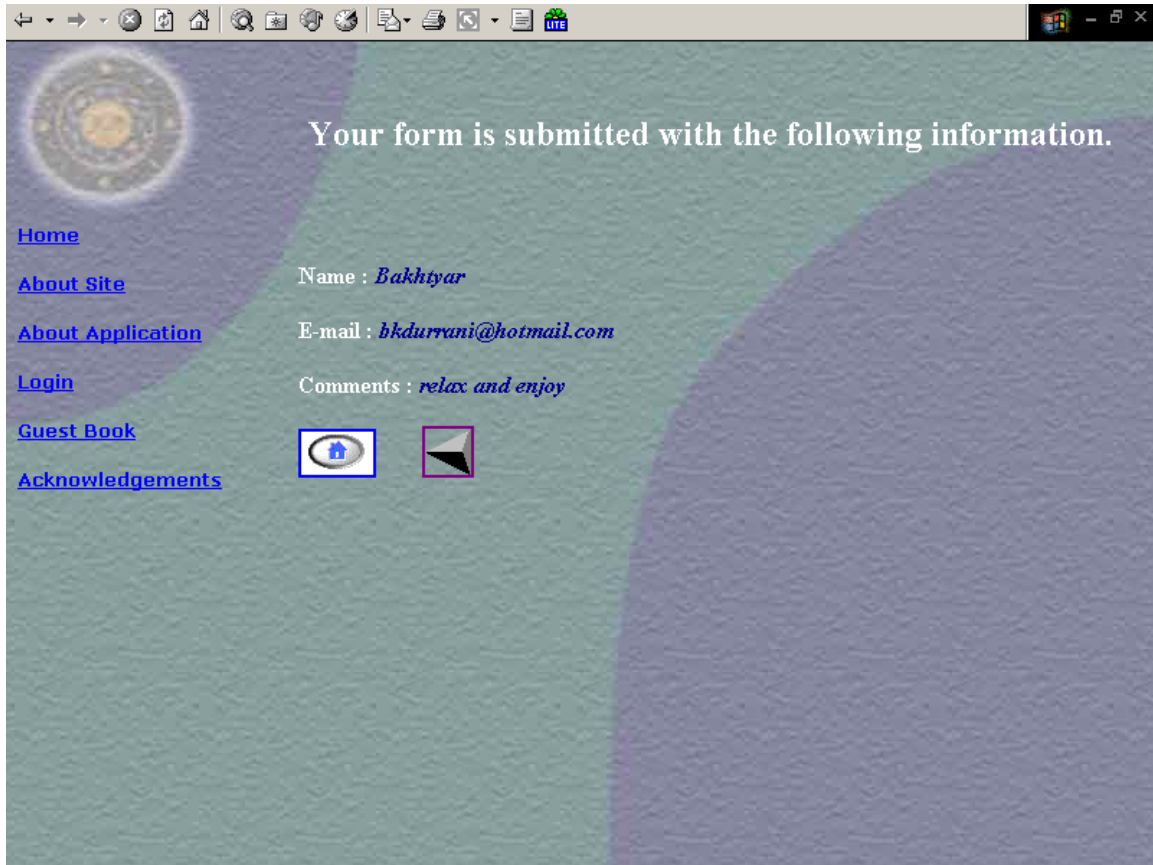
Name:

Email Address:

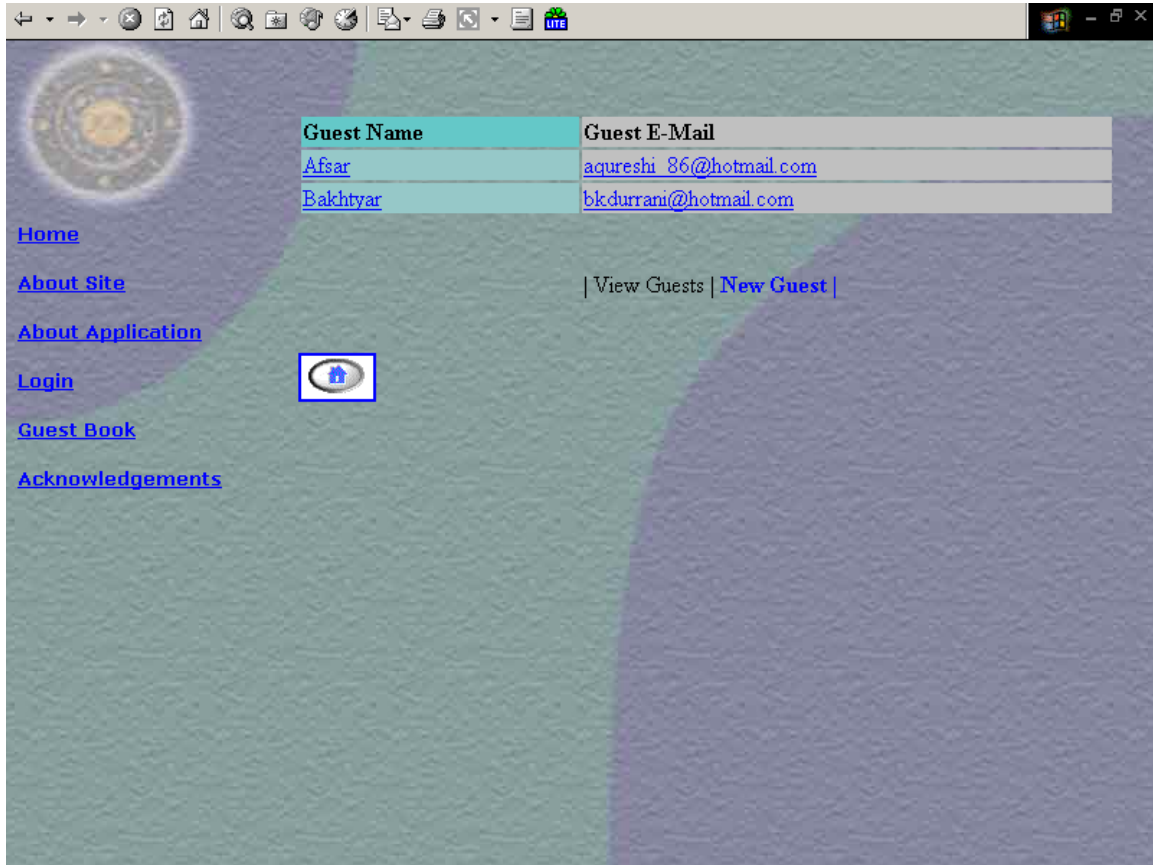
Comments:

| [View Guests](#) | [New Guest](#) |

19. Comments recorded



20. *Guests visited*




Guest Name	Guest E-Mail
Afsar	aquareshi_86@hotmail.com
Bakhtyar	bk.durrani@hotmail.com

[Home](#)

[About Site](#) | [View Guests](#) | [New Guest](#)

[About Application](#)

[Login](#) 

[Guest Book](#)

[Acknowledgements](#)

POP3

- a. Summary of QUIT command in the AUTHORIZATION state:

QUIT

Arguments: none

Restrictions: none

Possible Responses:

+OK

Examples:

C: QUIT

S: +OK dewey POP3 server signing off

- b. Summary of commands in TRANSACTION state:

STAT

Arguments: none

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK nn mm

Examples:

C: STAT

S: +OK 2 320

LIST [msg]

Arguments:

a message-number (optional), which, if present, may NOT refer to a message marked as deleted

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK scan listing follows
-ERR no such message

Examples:

C: LIST
S: +OK 2 messages (320 octets)
S: 1 120

S: 2 200
S: .
...
C: LIST 2
S: +OK 2 200
...
C: LIST 3
S: -ERR no such message, only 2 messages in maildrop

RETR msg**Arguments:**

a message-number (required) which may NOT refer to a message marked as deleted

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK message follows
-ERR no such message

Examples:

C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .

DELE msg**Arguments:**

a message-number (required) which may NOT refer to a message marked as deleted

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK message deleted
-ERR no such message

Examples:

C: DELE 1
S: +OK message 1 deleted
...
C: DELE 2
S: -ERR message 2 already deleted

NOOP

Arguments: none

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK

Examples:

C: NOOP
S: +OK

RSET

Arguments: none

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK

Examples:

C: RSET

S: +OK maildrop has 2 messages (320 octets)

c. Summary of QUIT command in UPDATE state:

QUIT

Arguments: none

Restrictions: none

Possible Responses:

+OK

-ERR some deleted messages not removed

Examples:

C: QUIT

S: +OK POP3 server signing off (maildrop empty)

...

C: QUIT

S: +OK POP3 server signing off (2 messages left)

d. Summary of optional commands:

TOP msg n

Arguments:

a message-number (required) which may NOT refer to a message marked as deleted, and a non-negative number of lines (required)

Restrictions:

may only be given in the TRANSACTION state

Possible Responses:

+OK top of message follows

-ERR no such message

Examples:

C: TOP 1 10

S: +OK

S: <the POP3 server sends the headers of the message, a blank line, and the first 10 lines

of the body of the message>
 S: .
 ...
 C: TOP 100 3
 S: -ERR no such message

UIDL [msg]

Arguments:

a message-number (optional), which, if present, may NOT refer to a message marked as deleted

Restrictions:

may only be given in the TRANSACTION state.

Possible Responses:

+OK unique-id listing follows
 -ERR no such message

Examples:

C: UIDL
 S: +OK
 S: 1 whqtswO00WBw418f9t5JxYwZ
 S: 2 QhdPYR:00WBw1Ph7x7
 S: .
 ...
 C: UIDL 2
 S: +OK 2 QhdPYR:00WBw1Ph7x7
 ...
 C: UIDL 3
 S: -ERR no such message, only 2 messages in maildrop

USER name

Arguments:

a string identifying a mailbox (required), which is of significance ONLY to the server

Restrictions:

may only be given in the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command

Possible Responses:

+OK name is a valid mailbox
 -ERR never heard of mailbox name

Examples:

C: USER frated
 S: -ERR sorry, no mailbox for frated here
 ...
 C: USER mrose
 S: +OK mrose is a real hoopy frood

PASS string

Arguments:

a server/mailbox-specific password (required)

Restrictions:

may only be given in the AUTHORIZATION state immediately
 after a successful USER command

Possible Responses:

+OK maildrop locked and ready
 -ERR invalid password
 -ERR unable to lock maildrop

Examples:

C: USER mrose
 S: +OK mrose is a real hoopy frood
 C: PASS secret
 S: -ERR maildrop already locked
 ...
 C: USER mrose
 S: +OK mrose is a real hoopy frood
 C: PASS secret
 S: +OK mrose's maildrop has 2 messages (320 octets)

APOP name digest

Arguments:

a string identifying a mailbox and a MD5 digest string
 (both required)

Restrictions:

may only be given in the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command

Possible Responses:

+OK maildrop locked and ready
-ERR permission denied

Examples:

S: +OK POP3 server ready
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK maildrop has 1 message (369 octets)

In this example, the shared secret is the string `tan-staaf'. Hence, the MD5 algorithm is applied to the string which produces a digest value of

c4c9334bac560ecc979e58001b3e22fb

e. Algorithm for Retrieving Mail

```
Client : +OK Server POP Ready!!
Client : USER xxx
Server : +OK
Client : PASS yyy
Server : +OK user successfully logged on
Client  : STAT
Server  : +OK n m
Client  : RETR 1
Server  : +OK
---{ data }-----
Client  : QUIT
Server  : +OK Server POP signing off
```

BIBLIOGRAPHY

1. [[RFC821](#)] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.
2. [[RFC822](#)] Crocker, D., "Standard for the Format of ARPA-Internet Text Messages", STD 11, [RFC 822](#), University of Delaware, August 1982.
3. [[RFC1321](#)] Rivest, R., "The MD5 Message-Digest Algorithm", [RFC 1321](#), MIT Laboratory for Computer Science, April 1992.
4. [[RFC1730](#)] Crispin, M., "Internet Message Access Protocol – Version 4", [RFC 1730](#), University of Washington, December 1994.
5. [[RFC1734](#)] Myers, J., "POP3 AUTHentication command", [RFC 1734](#), Carnegie Mellon, December 1994.
6. Agarwal , Tushar. SAPI Technical documentation, Digital GlobalSoft Limited, Hewlett-Packard Co., USA.
7. Agarwal , Tushar. MAPI Technical documentation, Digital GlobalSoft Limited, Hewlett-Packard Co., USA.
8. Agarwal , Tushar. TAPI Technical documentation, Digital GlobalSoft Limited, Hewlett-Packard Co., USA.