DESIGN AND IMPLEMENTATION OF WAP GATEWAY

SYNDICATE

SGT M.KASHIF IQBAL (Leader)
PC SYED ALI RASHID

PROJECT DS: **LEC ARIF RAZA**

Dissertation submitted for partial fulfillment of requirement of MCS/NUST for the award of the BE. Degree in Software Engineering.

Department of Computer Science
Military College of Signals
Rawalpindi

MAY 2002

Dedication

This report is dedicated to our parents who encouraged our studies and always stood beside us and supported and prayed for our success.

Acknowledgements

All thanks are due to Almighty Allah the most gracious who enabled us to complete this work .The author would like to acknowledge the highly valuable technical guidance from Lec Arif Raza who made this project possible for earliest completion. Mr. Ali Ahsan helped technically for the completion of this report. Technical help of project Manager at Hagler Bailey Mr. Mohannad, Mr Shahzad Haider from IBM Ace, Mr. Wasiq of R&DE is also highly appreciated. In the end we would also like to acknowledge Lt. Col. Farrukh for his efforts in giving us the concept of Computer Networks without which this project would not have completed.

Declaration

No portion of the work presented in this dissertation has been submitted in support of another award of qualification either at this institution or elsewhere

Abstract

This report details the design, development and testing of WAP Gateway. A WAP gateway is an implementation of WAP stack which translates between protocols in the WAP protocol suite and the TCP/IP protocol suite. This report is the documentation of a gateway which acts as a translator between Wireless Session Protocol (WSP) and Hypertext Transfer Protocol (HTTP). The report contains a description of the WAP protocol suite and method invocation in HTTP and WSP. The WAP Gateway is also responsible for the encoding of the WML content into binary representation so that minimal and compact data could be sent to the WAP Client. In the project only the connectionless part of the WAP protocol suite has been implemented. So in its present form the WAP Gateway works over an IP network using the UDP packets.

Table Of Contents

Chapter 1
Introduction

WAP –the wireless application protocol is a communication protocol and application environment for the deployment of information resources, advances telephony services and internet access from mobile devices . In recent years ,wireless telecommunication have become a common subject of technical papers. The new trend in technology is to provide users with the ability to have all they could possibly need in a pocket sized device. WAP is positioned at the convergence of two rapidly evolving network technologies, wireless data and the Internet.

1.1. Objective

The objective of our project was to design and implement a Wap gateway. The application implements the connectionless part of the wireless session protocol in the wap protocol stack, which mainly consists of method invocation facility. The WAP Gateway we have implemented performs two basic functions.

- Conversion between the WSP protocol and HTTP protocol.
- Encoding of WML content into binary format.

1.2. Demo Applications

We have also written some demo applications to show different kinds of applications you can use with your mobile phone. We have designed certain WML pages which show the working of the Gateway and also demonstrate the structure of wireless markup language. We have also implemented a servlet which plays an mp3 song selected by the WAP Client when the song is selected from a list of songs displayed to the user.

1.3. Skills

The syndicate has used a number of current technologies for the creation of the product. Some of these included what we had studied during the course of our studies and some skills needed to be learnt from scratch.

1.3.1. Applied Skills

The product required the skills of many of the modules studied throughout the course. Network communication understandings have been applied to create the end application as well as an understanding of programming language concepts. In addition, understandings of threads in a multi-threaded environment which have been previously studied have been applied to the successful development of the product. Good project planning is also a skill applied to produce a working system.

1.3.2. New Skills

The product required the understanding of Protocol Specification and design. Developing such an application has provided a new skill in application design. In addition to that we had to understand the concepts behind the WML language which was entirely new to us. The specifications of the HTTP protocol had also to be learnt. The wireless world was entirely a new field to us and we had to do a great research work to understand the concepts behind the WAP protocol.

1.4. Languages Used

In the implementation of the project we have used Java language. The reason behind this is that Java is platform independent. Secondly network programming is quite simple in java. Also java provides a great facility in dealing with threads. We have used the JAXP API for the parsing of WML documents. So our work of encoding of WML content into binary format was made quite simple by using this API. We have also used WML language in the applications design for the demonstration. WML is the language used by the WSP protocol to display the content. So its understanding and implementation was a must for us.

1.5. Software Interfaces

There are basically two other software interfaces to our project. One is the WAP Client and the other is the Web Server. We have used WAP IDE 3.1 as a simulator for the WAP Client and Web Server we have used is Lite Web Server 2.2.1 which was also used by us during our course of Java language.

1.6. Environment

We have tested our application on Windows 98, Windows 2000, Windows XP and also on WINNT. As the implementation language is Java so there is no problem of platform.

Chapter 2
Wireless Access Protocol

2.1. Why WAP?

Both the wireless data market and the Internet are growing very quickly and are continuously reaching new customers. The explosive growth of the Internet has fuelled the creation of new and exciting information services. Most of the technology developed for the Internet has been designed for desktop and larger computers and medium to high bandwidth, generally reliable data networks. Mass-market, hand-held wireless devices present a more constrained computing environment compared to desktop computers. Because of fundamental limitations of power and form-factor, mass-market handheld devices tend to have:

- Less powerful CPUs,
- Less memory (ROM and RAM),
- Restricted power consumption,
- Smaller displays, and
- Different input devices (e.g., a phone keypad).

Similarly, 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, and
- Less predictable availability.

For example if we are going to allow Internet access from a mobile phone, we first need to take into account these limitations of the client device .the internet protocols are far from being suitable for use with mobile phone communication. They introduce far too many overheads , requiring many messages between clients and server just to set up a connection. These overheads call fro high processing power on the client device. Similarly most of the existing content on the internet on the internet is in html form and due to transport limitations and bandwidth constraints it becomes inappropriate to transfer such content on a wireless device. It is due to these severe limitations, a new set of protocols more appropriate to communication with the wireless devices is needed. It is here the wap technology comes to the rescue. WAP, short for Wireless Application Protocol, is a collection of languages and tools and an infrastructure for implementing services for mobile phones. Traditionally such services have worked via normal phone calls or short textual messages (e.g., SMS messages in GSM networks). Neither are very efficient to use, nor very user friendly. WAP makes it possible to implement services similar to the World Wide Web.

2.2. WAP Forum

The Wireless Application Protocol (WAP) is a result of the WAP Forum's efforts to promote industry-wide specifications for technology useful in developing applications and services that operate over wireless communication networks. WAP specifies an application framework and network protocols for wireless devices
such as mobile telephones, pagers, and personal digital assistants (PDAs). The specifications extend and leverage mobile networking technologies (such as digital data networking standards) and Internet technologies (such as XML, URLs, scripting, and various content formats). The effort is aimed at enabling operators, manufacturers, and content developers to meet the challenges in building advanced differentiated services and implementations in a fast and flexible manner.

The objectives of the WAP Forum are:

- To bring Internet content and advanced data services to digital cellular phones and other wireless terminals.

- To create a global wireless protocol specification that will work across differing wireless network technologies.

- To enable the creation of content and applications that scale across a very wide range of bearer networks and device types.

- To embrace and extend existing standards and technology wherever appropriate.

- The WAP Architecture Specification is intended to present the system and protocol architectures essential to achieving the objectives of the WAP Forum. The WAP specifications address mobile network characteristics and operator needs by adapting existing network technology to the special requirements of mass-market, hand-held wireless data devices and by introducing new technology where appropriate.

Wap forum created by phone.com, Ericsson, Motorola and Nokia shared there knowledge and the partnership soon evolved into the now all encompassing WAP SPECIFICATIONS that include complementary application session transaction, security and transport protocol layers. A new markup language called the Wireless Markup Language (WML) has also been created. These protocols minimize the problems outlined above associated with the use of internet protocols for wireless data transfer. They do this by eliminating unnecessary data transfers and using binary code to reduce the amount of data that has to be sent. Also wireless sessions are designed to be easily suspended and resumed , without the connection overheads associated with the internet protocols. Thus the protocols are well suited to the low bandwidth associated with the wireless communication.

The standardization of methods to access the internet through the mobile phones has brought many benefits to many different people. For the end users a breadth of choice of devices networks and applications a has arisen in the competitive market since the specifications are not biased towards any company. The network operators have been able to extend there customer base as they can offer newer services independent of the network used .for the service providers there are new functionalities such as push technologies and WTA .

2.3. The Business Perspective

As the new standard protocol for providing content to wireless devices, wap has been accepted on the telecommunication market with enthusiasm from all sides, as the growth in the stock market of some of the companies involved with wap can confirm the high penetration rates of the mobile phones across Europe and America mean that mobile commerce has become significant .Many businesses were caught by surprise by the rapid rise of e commerce and so have jumped quickly to supply WAP services in an effort not to be left behind.

Application developers play an important role in the new born Wap industry . in addition to providing Value Added Services there is a strong demand for services that are available on the web to be ported to WAP. On of the major advantages to the WAP is that it's markup language WML, is based on XML .this effectively means that it should be easier to provide content in a device independent way.

As with all new technologies, the expectations of WAP were very high when it was introduces . before the wap phones were available ,everyone was expecting to surf the internet from there mobile phones like a normal browser but the reality is quiet different .Wap is intended to provide a common application environment for mobile devices and its protocols are based on the internet protocols. However this does not mean that wap was devised with the intent of porting the entire content of the internet to the mobile devices.

2.4. WAP Application Architecture

The wap protocols were deigned with the web protocols in mind. The goal of the wap was to use the underlying web structure but to render communication between content providers and mobile devices more efficient and less time consuming than if the web protocols themselves were used.

Since the wap architecture   has been designed closely to follow the web, the client server paradigm used by the internet has been inherited by WAP. The main difference, however is the presence of the wap gateway for translating between HTTP and WAP.

Before continuing we will take a closer look at these technical terms.

2.4.1. WAP Device:

This term indicates the physical device that you use to access WAP applications and content. It doesn't necessarily have to be a mobile phone-it might be a PDA or a hand held computer. More generally it's a wap compliant device.

2.4.2. WAP Client:

In a network environment, a client is typically the logical entity that is operated by the user and communicates with the server entity. In the wap world the client is the entity that receives the content.

2.4.3. User Agent:

An agent is normally the software that deals with protocols and wap is no exception to this. The wap client contains two different agents.

- The WAE user agent
- The WTA user agent.

2.4.4. WAP Gateway:

This is the element that sits logically between the wap device and the origin server. It acts as an interpreter between the two enabling them to communicate with the wireless operator network but you can install your own gateway.

2.4.5. Network Operator:

This is the company or the organization that provides carrier services to its subscribers. As an example, the company you are paying your telephone bills to is your network operator .A network operator enables you to make calls to other phones from your telephone and in addition, provides you with different services such as voice mail.

2.4.6. Content /Application Server:

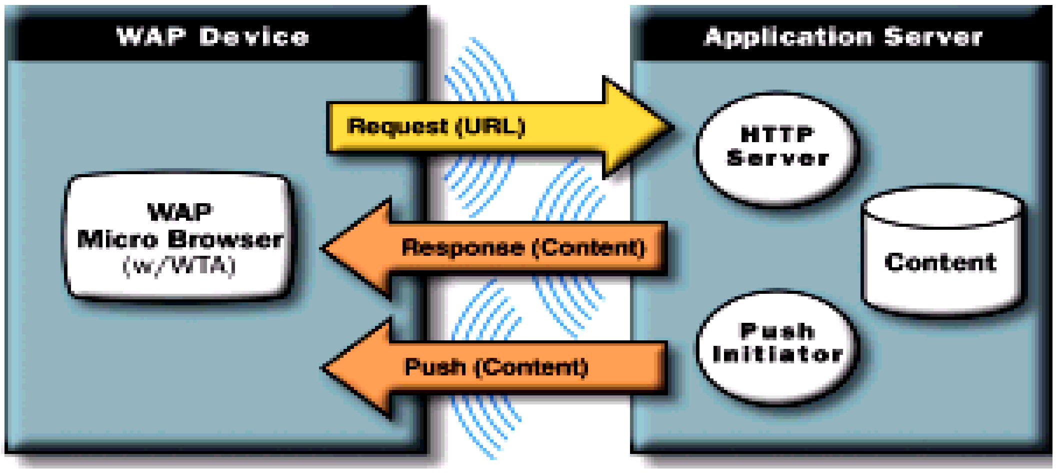
This is the element that hosts the internet content that is sent to the clients when they make a request for it. A web server is an origin server, providing html content.

Chapter 3
WAP Architecture & Internal Structure

3.1. Architecture Overview

3.1.1. The World-Wide Web Model

The Internet World-Wide Web (WWW) architecture provides a very flexible and powerful programming model. Applications and content are presented in standard data formats, and are browsed by applications known as web browsers. The web browser is a networked application, i.e., it sends requests for named data objects to a network server and the network server responds with the data encoded using the standard formats.



World-Wide Web Programming Model

The WWW standards specify many of the mechanisms necessary to build a general-purpose application environment, including:

- Standard naming model – All servers and content on the WWW are named with an Internet-standard Uniform Resource Locator (URL).

- Content typing – All content on the WWW is given a specific type thereby allowing web browsers to correctly process the content based on its type.

- Standard content formats – All web browsers support a set of standard content formats. These include the Hypertext Markup Language (HTML) [HTML4], the JavaScript scripting language [ECMAScript, JavaScript], and a large number of other formats.

- Standard Protocols – Standard networking protocols allow any web browser to communicate with any web server. The most commonly used protocol on the WWW is the HyperText Transport Protocol (HTTP).

This infrastructure allows users to easily reach a large number of third-party applications and content services. It also allows application developers to easily create applications and content services for a large community of clients.

The WWW protocols define two classes of servers:

3.1.1.1. Origin server

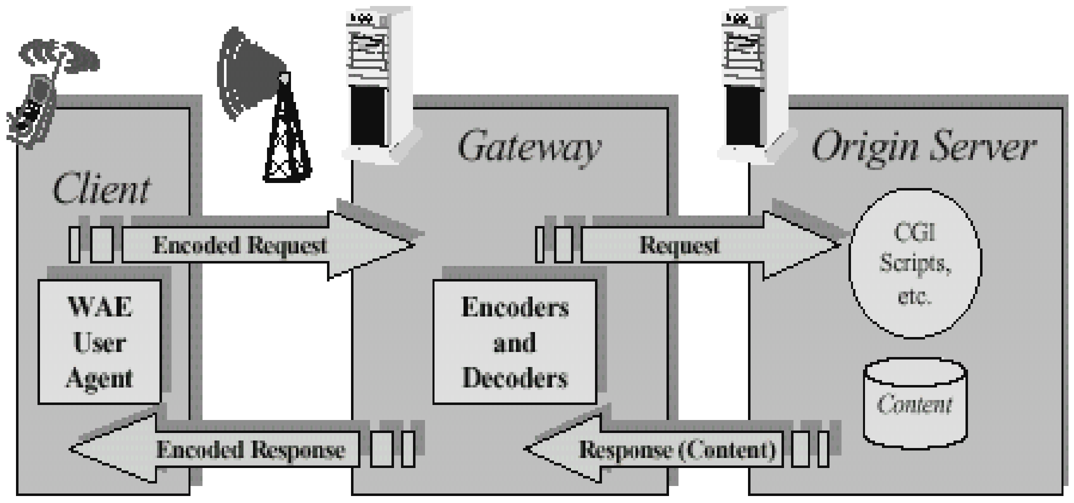
The server on which a given resource (content) resides or is to be created.

3.1.1.2. Proxy

This is an intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. The proxy typically resides between clients and servers that have no means of direct communication, eg across a firewall. Requests are either serviced by the proxy program or passed on, with possible translation, to other servers. A proxy must implement both the client and server requirements of the WWW specifications.

3.2. The WAP Model

The WAP programming model is similar to the 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 (eg, 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 the WAP technology.



WAP Programming Model (fig1)

WAP content and applications are specified in a set of well-known content formats based on the familiar WWW content formats. Content is transported using a set of standard communication protocols based on the WWW communication protocols. A micro browser in the wireless terminal co-ordinates the user interfaces and is analogous to a standard web browser.

WAP defines a set of standard components that enable communication between mobile terminals and network servers, including:

3.2.1. Standard naming model

WWW-standard URLs are used to identify WAP content on origin servers. They are also used to identify local resources in a device, eg call control functions.

3.2.2. Content typing

All WAP content is given a specific type consistent with WWW typing. This allows WAP user agents to correctly process the content based on its type.

3.2.3. Standard content formats

WAP content formats are based on WWW technology and include display markup, calendar information, electronic business card objects, images and scripting language.

3.2.4. Standard communication protocols

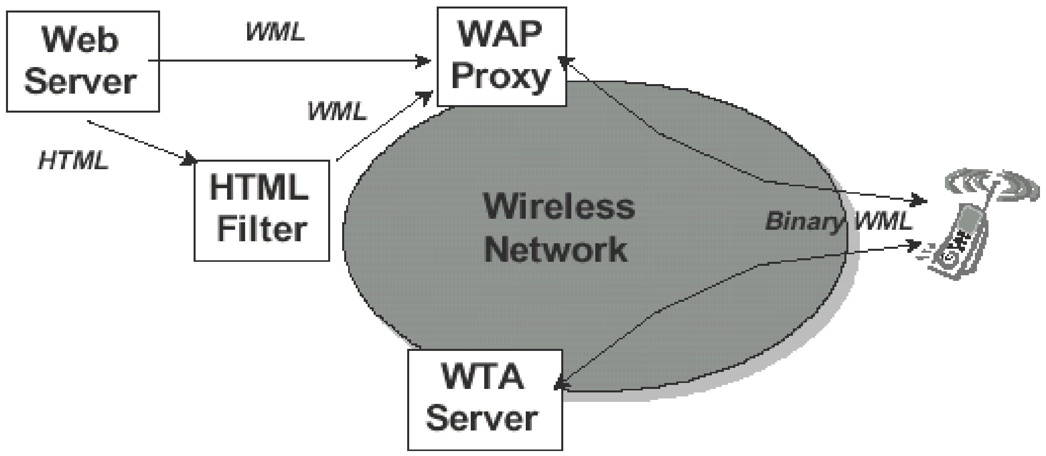
WAP communication protocols enable the communication of browser requests from the mobile terminal to the network web server. The WAP content types and protocols have been optimized for mass market, hand-held wireless devices. WAP utilizes proxy technology to connect between the wireless domain and the WWW. The WAP proxy typically is comprised of the following functionality:

3.2.4.1. Protocol Gateway

The protocol gateway translates requests from the WAP protocol stack (WSP, WTP, WTLS, and WDP) to the WWW protocol stack (HTTP and TCP/IP).

3.2.4.2. Content Encoders and Decoders

The content encoders translate WAP content into compact encoded formats to reduce the size of data over the network. This infrastructure ensures that mobile terminal users can browse a wide variety of WAP content and applications, and that the application author is able to build content services and applications that run on a large base of mobile terminals. The WAP proxy allows content and applications to be hosted on standard WWW servers and to be developed using proven WWW technologies such as CGI scripting. While the nominal use of WAP will include a web server, WAP proxy and WAP client, the WAP architecture can quite easily support other configurations. It is possible to create an origin server that includes the WAP proxy functionality. Such a server might be used to facilitate end-to-end security solutions, or applications that require better access control or a guarantee of responsiveness, eg, WTA.



Example Wap Network

In the example, the WAP client communicates with two servers in the wireless network. The WAP proxy translates WAP requests to WWW requests thereby allowing the WAP client to submit requests to the web server. The proxy also encodes the responses from the web server into the compact binary format understood by the client.

If the web server provides WAP content (e.g., WML), the WAP proxy retrieves it directly from the web server. However, if the web server provides WWW content (such as HTML), a filter is used to translate the WWW content into WAP content. For example, the HTML filter would translate HTML into WML.

The Wireless Telephony Application (WTA) server is an example origin or gateway server that responds to requests from the WAP client directly. The WTA server is used to provide WAP access to features of the wireless network provider's telecommunications infrastructure.

Before we look at the detail of how are the Wap protocols are structured let us briefly examine the definitions of a protocol and a layer.

3.3. Protocols

As anyone who has done any international traveling knows it is quite important when you travel to adapt your clothing and behavior to the place you are in. It is also important to speak a common language that allows others to understand what you are saying. The same problem arises with the telecommunication networks. There are many different devices and networks and to allow them to communicate with each other you must provide them with a common language. Protocols are the answers to this problem. There are lots of different kinds from very simple ones to elaborate ones but they all have the same property in common. They allow the computers to communicate with each other.

A protocol defines the type and the structure of messages that two devices have to use when they are communicating with each other

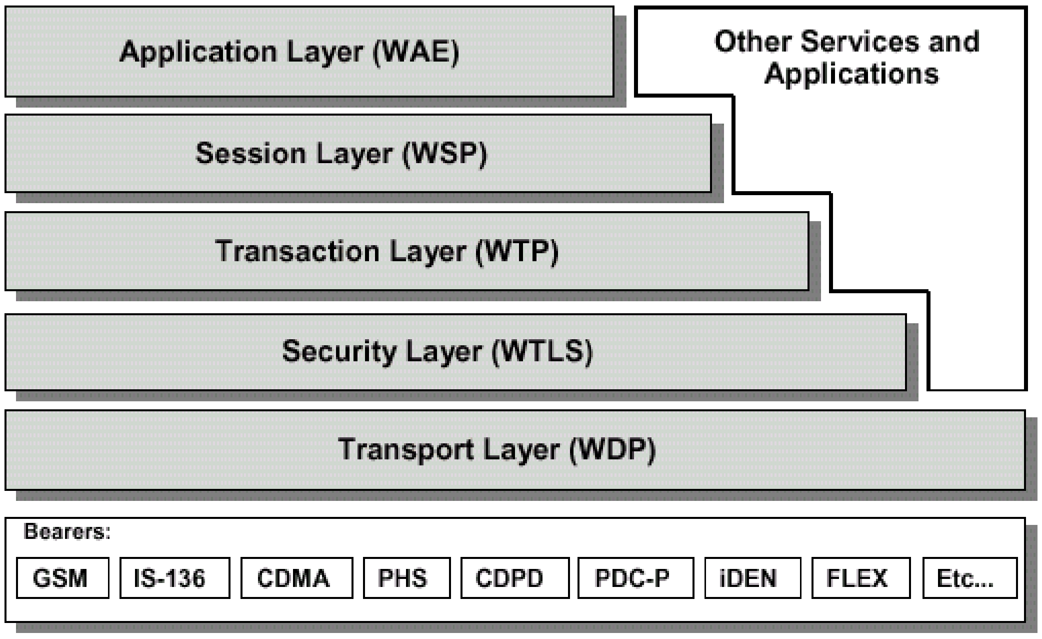
3.4. Layers

Since the protocols are functionally and logically divided into different groups of functionality, they are also physically formed into layers each one providing a specific service to the next layer. One layer may provide methods to send bits down the physical layer; another may supply methods to establish a connection. The protocol stack is the set of all layers that compose the set of protocols.

3.5. Components of the WAP Architecture

3.5.1. WAP Protocol Stack

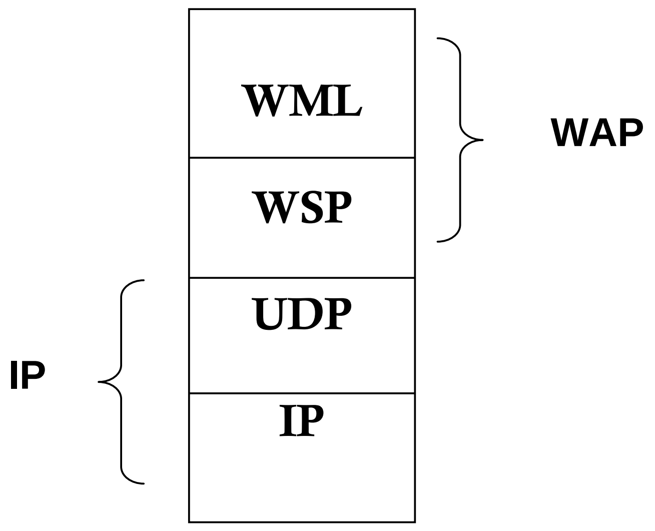
The WAP architecture provides a scaleable and extensible environment for application development for mobile communication devices. This is achieved through a layered design of the entire protocol stack . Each of the layers of the architecture is accessible by the layers above, as well as by other services and applications.



WAP Architecture

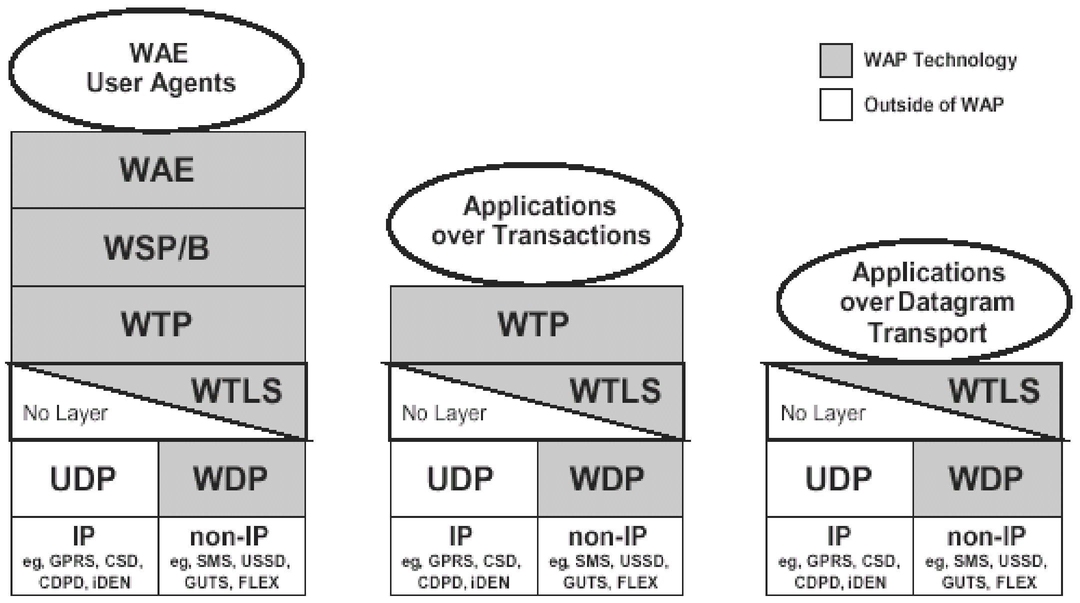
3.5.2. Sample Configurations of WAP Technology

The suite is designed so that one can take any consecutive set of the WAP layers and use them in an already existing framework. For instance, when browsing on the Internet using a WAP client one could use the protocol stack in the f where the two upper protocols are from the WAP suite and the lower two are from the IP suite.



Example of a WAP stack when browsing with a WAP client

WAP technology is expected to be useful for applications and services beyond those specified by the WAP Forum. The next figure depicts several possible protocol stacks using WAP technology. These are for illustrative purposes only and do not constitute a statement of conformance or interoperability.


WAP Technology
Outside of WAP
WAE User Agents
WAE
WSP/B
WTP
WTLS
No Layer
UDP
WDP
IP eg, GPRS, CSD, CDPD, iDEN
non-IP eg, SMS, USSD, GUTS, FLEX
Applications over Transactions
Applications over Datagram Transport

Sample WAP Stacks

The leftmost stack represents a typical example of a WAP application, i.e., WAE user agent, running over the complete portfolio of WAP technology. The middle stack is intended for applications and services that require transaction services with or without security. The rightmost stack is intended for applications and services that only require datagram transport with or without security.

3.5.3. Description of WAP Stack

The WAP stack was derived from and inherited most of the characteristics of, the ISO OSI reference model. The main difference between the two is the no of layers. The following sections provide a description of the various elements/ layers of the protocol stack architecture.

3.6. Application Layer:

WAE (Wireless Application Environment) provides an application environment intended for development and execution of portable applications and services. The Wireless Application Environment (WAE) is a general-purpose application environment based on a combination of World Wide Web (WWW) and Mobile Telephony technologies. The primary objective of the WAE effort is to establish an interoperable environment that will allow operators and service providers to build applications and services that can reach a wide variety of different wireless platforms in an efficient and useful manner. WAE includes a micro-browser environment containing the following functionality:

- Wireless Markup Language (WML) – a lightweight markup language, similar to HTML, but optimized for use in hand-held mobile terminals;

- WMLScript – a lightweight scripting language, similar to JavaScript.

- Wireless Telephony Application (WTA, WTAI) – telephony services and programming interfaces and· Content Formats – a set of well-defined data formats, including images, phone book records and calendar information.

3.6.1. WAE Architecture Overview

The WAE architecture includes all elements of the WAP architecture related to application specification and execution. At this point, the WAE architecture is predominately focused on the client-side aspects of WAP's system architecture, namely items relating to user agents. Specifically, the WAE architecture is defined primarily in terms of networking schemes, content formats, programming languages and shared services. Interfaces are not standardized and are specific to a particular implementation. This approach allows WAE to be implemented in a variety of ways without compromising interoperability or portability. This approach has worked particularly well with a browser (a class of user agents) model such as that used in the World-Wide-Web (WWW). The Internet and the WWW are the inspiration and motivation behind significant parts of the WAE specification, and consequently, a similar approach is used within WAE.

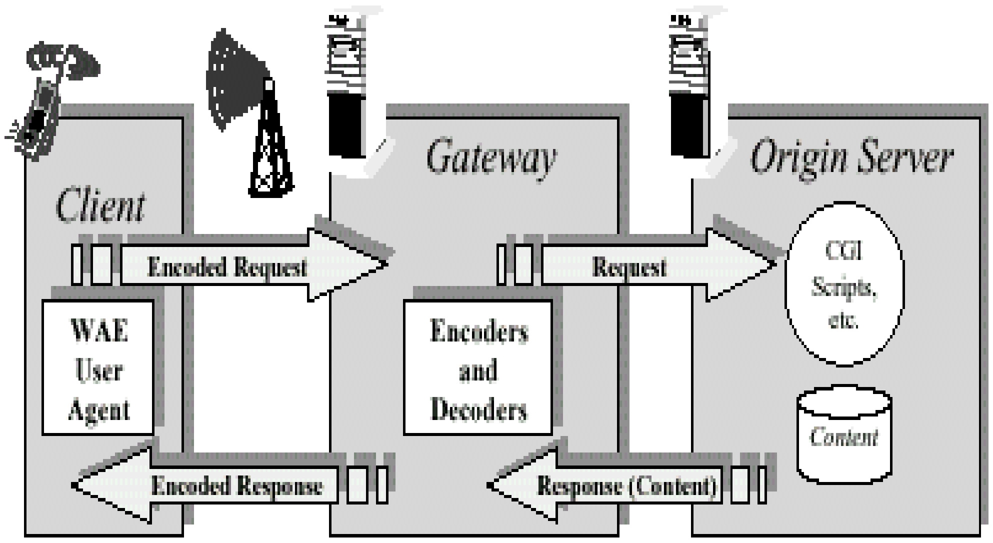
3.6.2. The WAE Model

WAE adopts a model that closely follows the WWW model. All content is specified in formats that are similar to the standard Internet formats. Content is transported using standard protocols in the WWW domain and an optimized HTTP-like protocol in the wireless domain. WAE has borrowed from WWW standards including authoring and publishing methods wherever possible. The WAE architecture allows all content and services to be hosted on standard Web origin servers that can be incorporate proven technologies (eg, CGI). All content is located using WWW standard URLs.

The WAE does not specify how a user agent should display information, e.g. a WML deck, nor does it dictate (or even assume) a man-machine interface. It's up to the vendors of user agents to solve these issues in any way they like.

An important part of the WAE is the WAP gateway. Its main objectives are:

- Minimize data sent over the air
- Minimize the computational energy required by the client to process the data



3.6.3. Elements of the WAE model

3.6.3.1 WAE User Agents

It is basically the client-side in-device software that provides specific functionality (eg, display content) to the end-user. User agents (such as browsers) are integrated into the WAP architecture. They interpret network content referenced by a URL. WAE includes user agents for the two primary standard contents: encoded Wireless Markup Language (WML) and compiled Wireless Markup Language Script (WMLScript.)

3.6.3.2. Content Generators

Applications (or services) on origin servers (eg, CGI scripts) that produce standard content formats in response to requests from user agents in the mobile terminal. WAE does not specify any standard content generators but expects that there will be a great variety available running on typical HTTP origin servers commonly used in WWW today.

.

3.6.3.3. Standard Content Encoding

A set of well-defined content encoding, allowing a WAE user agent (eg, a browser) to conveniently navigate web content. Standard content encoding includes compressed encoding for WML, byte code encoding for WMLScript, standard image formats, a multi-part container format and adopted business and calendar data formats.

3.7. Wireless Telephony Applications (WTA)

A collection of telephony specific extensions for call and feature control mechanisms that provide authors (and ultimately end-users) advanced Mobile Network Services.
The user agent characteristics are communicated to the server via standard capability negotiation mechanisms that allows applications on the origin server to determine characteristics of the mobile terminal device. WAE defines a set of user agent capabilities that will be exchanged using WSP mechanisms. These capabilities include such global device characteristics as WML version supported, WMLScript version supported, floating-point support, image formats supported and so on.

WAE architecture relies heavily on WWW's URL and HTTP semantics. WAE assumes:

- The existence of a generalized architecture for describing gateway behavior for different types of URLs and
- Support for connection to at least one WAP gateways.

WAE is based on the architecture used for WWW proxy servers. The situation where a user agent (eg, a browser) must connect through a proxy to reach an origin server (i.e., the server that contains the desired content) is very similar to the case of a wireless device accessing a server through a gateway.
Most connections between the browser and the gateway use WSP, regardless of the protocol of the destination server.

The URL, used to distinguish the desired content, always specifies the protocol used by the destination server regardless of the protocol used by the browser to connect to the gateway. In other words, the URL refers only to the destination server's protocol and has no bearing on what protocols may be used in intervening connections.

In addition to performing protocol conversion by translating requests from WSP into other protocols and the responses back into WSP, the gateway also performs content conversion. This is analogous to HTML/HTTP proxies available on the Web today. For example, when an HTTP proxy receives an FTP or Gopher directory list, it converts the list into an HTML document that presents the information in a form acceptable to the browser. This conversion is analogous to the encoding of content destined to WAE user agents on mobile devices.

3.7.1. HTTP

The browser, in this case, communicates with the gateway using WSP. The gateway in turn would provide protocol conversion functions to connect to an HTTP origin server.

As an example, a user, with a WAP-compliant telephone, requests content using a specific URL. The telephone browser connects to the operator-controlled gateway with WSP and sends a GET request with that URL. The gateway resolves the host address specified by the URL and creates an HTTP session to that host. The gateway performs a request for the content specified by the URL. The HTTP server at the contacted host processes the request and sends a reply (eg, the requested content). The gateway receives the content, encodes it, and returns it to the browser

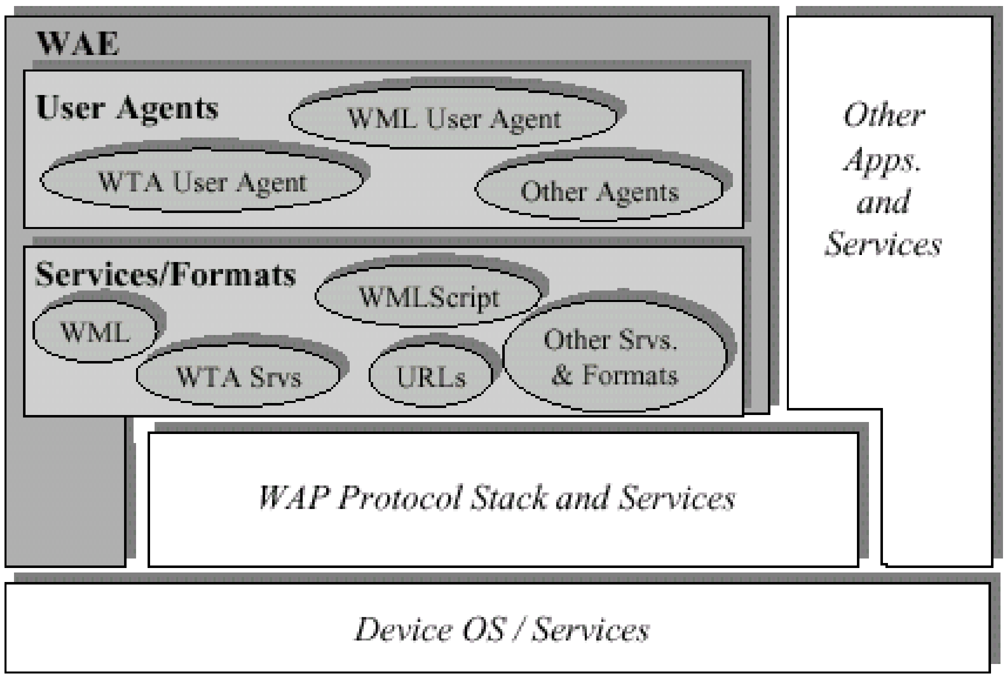
3.7.2. Components of WAE

As illustrated in the following Figure, WAE is divided into two logical layers:

- User agents, which includes such items as browsers, phonebooks, message editors, etc; and

- Services and Formats, which include common elements and formats accessible to user agents such as WMLWMLScript, image formats, vCard and vCalendar formats, etc.

WAE separates services from user agents and assumes an environment with multiple user agents. This logical view, however, does not imply or suggest an implementation. For example, WAE implementations may choose to combine all the services into a single user agent. Others, on the other hand, may choose to distribute the services among several user agents. The resulting structure of a WAE implementation is determined by the design decisions of its implementers and should be guided by the specific constraints and objectives of the target environment.



3.7.2.1. WAE User Agents

The WML user agent is a fundamental user agent of the WAE. However, WAE is not limited to a WML user agent. WAE allows the integration of domain-specific user agents with varying architectures and environments. In particular, a Wireless Telephony Application (WTA) user agent has been specified as an extension to the WAE specification for the mobile telephony environments. The WTA extensions allow authors to access and interact with mobile telephone features (eg, call control) as well as other applications assumed on the telephones, such as phonebooks and calendar applications

3.7.2.2. WAE Services and Formats

The WAE Services and Formats layer includes the bulk of technical contribution of the WAE effort. The following section provides an overview of the major components of WAE including the Wireless Markup Language (WML), the Wireless Markup Scripting language (WMLScript), WAE applications and WAE supported content formats.

3.7.3. WML

WML is a tag-based document language. In particular, it is an application of a generalized mark-up language. WML shares a heritage with the WWW's HTML[HTML4] and Handheld Device Markup Language [HDML2]. WML is specified as an XML [XML] document type. It is optimized for specifying presentation and user interaction on limited capability devices such as telephones and other wireless mobile terminals. WML and its supporting environment were designed with certain small narrow-band device constraints in mind including small displays, limited user-input facilities, narrow band network connections, limited memory resources and limited computational resources. Given the wide and varying range of terminals targeted by WAP, considerable effort was put into the proper distribution of presentation responsibility between the author and the browser implementation.

WML is based on a subset of HDML version 2.0 [HDML2]. WML changes some elements adopted from HDML and introduces new elements, some of which have been modeled on similar elements in HTML. The resulting WML implements a card and deck metaphor. It contains constructs allowing the application to specify documents made up of multiple *card*s. An interaction with the user is described in a set of cards, which can be grouped together into a document (commonly referred to as a *dec*k). Logically, a user navigates through a set of WML cards. The user navigates to a card, reviews its contents may enter requested information, may make choices, and then moves on to another card. Instructions imbedded within cards may invoke services on origin servers as needed by the particular interaction. Decks are fetched from origin servers as needed. WML decks can be stored in 'static' files on an origin server, or they can be dynamically generated by a content generator running on an origin server. Each card, in a deck, contains a specification for a particular user interaction.

WAE has been designed using the Internet and the World Wide Web as its main template. When designing WML the designers chose to make it a subset of XML (eXtensible Markup Language). The main reason was that the data would be easier separated from the markup tags and it would also be easier to generate WML cards, for instance from data extracted from a database. In an XML document a tag must have an ending tag. Between the two tags one may introduce other tag-pairs. This way an XML document may be considered being a tree and information is extracted by just traversing the tree. XML specification only specifies *how* XML tags shall be used but *not which* tags you should use. To construct a language using XML tags one must define which tags one want to use and exactly specify which attributes the authors may use. This is done by including a document type declaration (DTD) into the XML document.

WML doesn't specify how implementations (e.g. cellular phones) request input from a user. Instead it specifies the *intent* in an abstract way. The language is man-machine interface independent and it's up to the vendors to make their own interface. WML supports UNICODE characters making it possible to display information in almost any language.

Although WML has limited capabilities when compared to HTML, it has never the less a wide range of features.

3.7.3.1. Support for Text

When including text in a card , the programmer can use emphasis elements .One should remember however that the features each browser implements may vary and some do not support tables.

3.7.3.2. Support for Images

A new format has been created for displaying images called WBMP . Images compliant with this new standard are currently black and white.

3.7.3.3. Navigation and History Stack

Common navigation and history functionalities are also included.

3.7.4. WMLSCRIPT

On the World Wide Web, JavaScripts are widely used to produce web pages with more ``intelligence''. For instance, a form may be checked to see if the input is formatted correctly or one can make a small computation by clicking at a hyperlink. WAE defines it's own scripting language called WMLScript. WMLScript is a weakly typed scripting language and it's based entirely on JavaScript. By supplying a scripting language WAE applications may use procedural logic in the WML decks they produce and also let the interaction between the client and the server be based on events. As WML, WMLScript supports UNICODE and international characters. WMLScript provides the application programmer with a variety of interesting capabilities:

- The ability to check the validity of user input before it is sent to the content server.
- The ability to access device facilities and peripherals.
- The ability to interact with the user without introducing round-trips to the origin server (eg, display an error message).

WMLScript supports several categories of operations such as assignment operations, arithmetic operations, logical operations and comparison operations. WMLScript supports several categories of functions including Local script functions (i.e., script functions defined inside the same script that the calling expression is in), External script functions (i.e., script functions defined in another script not containing the calling expression) and Standard library functions (i.e., functions defined in a library that is part of the WAE specification.) WMLScript defines several standard libraries including a language library, a string library, a browser library, a floating point library and a dialog library.

3.7.5. Wireless Binary XML

One important task of the WAP Gateway is to reduce the amount of data sent over the air. If an XML document is sent to a WAP client, the document is compiled into a compact binary form of XML named Wireless Binary XML (WBXML). This compilation is basically done by replacing the WML tags with one-byte tokens and removing all comments from the document. A special case of an XML document is a WML deck. The deck is compiled into a binary representation by the gateway before being sent to the WAP client.

3.7.6. Wireless Telephony Application

WTA is a collection of telephony specific extensions for call and feature control mechanisms that make advanced Mobile Network Services available to authors and end-users. WTA merges the features and services of data networks with the services of voice networks. It introduces mechanisms that ensure secure access to important resources within mobile devices. The WTA framework allows real-time processing of events important to the end-user while browsing. Within the WTA framework, the client and server co-ordinate the set of rules that govern event handling via an event table. WTA origin servers can adjust the client’s rules by pushing (or updating) a client’s event table if required as defined in WTA.

The Wireless Telephony Application Framework has four main goals:

- Enable Network Operators to provide advanced telephony services that are well integrated and have consistent user interfaces.

- Enable Network Operators to create customized content to increase demands and accessibility for various services in their Networks.

- Enable Network Operators to reach a wider range of devices by leveraging generic WAE features that allow the operator to create content independent of device specific characteristics and environments.

- · Enable third party developers to create network-independent content that access basic features (i.e., non-privileged). Most of the WTA functionality is reserved for the Network Operators, as in-depth knowledge and access to the mobile network are needed to fully take advantage of the mobile network's features.

Wireless Telephony Application (WTA) is an application framework for telephony services. WTA user agents are able to make calls and edit the phone book by calling special WMLScript functions or by accessing special URLs. This way, if one writes WML decks containing names of people and their phone numbers you may add them to your phone book or call them right away just by clicking the appropriate hyperlink on the screen.

The WTA framework also includes WTA servers. A WTA server can be thought of as a web server with the ability to interact with the mobile network and other entities (e.g. a voice mail system). The services on a WTA server are addressed by URLs. For instance, one could write an URL which tells the WTA server to contact the voice mail system and allows users to listen to their voice mail. WTA uses a WSP session, which is called the WTA session for communication between the client and the server. A WTA user-agent may have one or many WTA sessions simultaneously.

3.8. Session Layer

The session layer protocol of the WAP suit is called the Wireless Session Protocol (WSP) and is defined in Wireless Session Protocol Specification. WSP also includes some features not included in the HTTP protocol. These features are there because of the mobile nature of the WAP clients. For example, a client, e.g. a mobile phone, should not loose its connection to the server when it changes base stations.

WSP (Wireless session Protocol) supplies methods for the organized exchange Of content between client /server applications. The Session layer protocol family in the WAP architecture is called the Wireless Session Protocol, WSP. WSP provides the upper-level application layer of WAP with a consistent interface for two session services. The first is a connection-mode service that operates above a transaction layer protocol WTP, and the second is a connectionless service that operates above a secure or non-secure datagram transport service. The Wireless Session Protocols currently offer services most suited for browsing applications (WSP/B). WSP/B provides HTTP 1.1 functionality and incorporates new features such as long-lived sessions, a common facility for data push, capability negotiation and session suspend/resume. The protocols in the WSP family are optimized for low-bandwidth bearer networks with relatively long latency.

3.8.1. Basic Types & Functionality

The core of the WSP/B design is a binary form of HTTP. Consequently the requests sent to a server and responses going to a client may include both headers (meta-information) and data. All the methods defined by HTTP/1.1 are supported. In addition, capability negotiation can be used to agree on a set of extended request methods, so that full compatibility to HTTP/1.1 applications can be retained.

WSP/B provides typed data transfer for the application layer. The HTTP/1.1 content headers are used to define content type, character set encoding, languages, etc, in an extensible manner. However, compact binary encodings are defined for the well-known headers to reduce protocol overhead. WSP/B also specifies a compact composite data format that provides content headers for each component within the composite data object. This is a semantically equivalent binary form of the MIME “multipart/mixed” format used by HTTP/1.1.

WSP/B itself does not interpret the header information in requests and replies. As part of the session creation process, request and reply headers that remain constant over the life of the session can be exchanged between service users in the client and the server. These may include acceptable content types, character sets, languages, device capabilities and other static parameters. WSP/B will pass through client and server session headers as well as request and response headers without additions or removals.

The lifecycle of a WSP/B session is not tied to the underlying transport. A session can be suspended while the session is idle to free up network resources or save battery. A lightweight session re-establishment protocol allows the session to be resumed without the overhead of full-blown session establishment. A session may be resumed over a different bearer network.

The Wireless Session Protocol enables services to exchange data between applications in an organized way. It includes two different protocols.

1. Connection oriented session services – operates over Wireless Transaction Protocol (WTP)
2. Connectionless Session Services- operates directly over the Wireless Transport Layer

Session Services are those functionalities that help to set up a connection between a client and a server. A service is delivered through the use of primitives it provides. Primitives are defined messages a client sends to the server to request a service facility. In WSP, for example one of the primitives is S-connect in which we can request the creation of a connection to the server.

The connection oriented session service provides facilities used to manage a session and to transmit reliable data between the client and the connection oriented session service provides facilities used to manage a session and to transmit reliable data between a client and a server .The session created can be then suspended and resumed later if the transmission of data becomes impossible also when the push technology takes off unsolicited data can be pushed from the server to the client in confirmed or unconfirmed way .In confirmed push the server is notified upon reception of the pushed data by the client .

In unconfirmed push the server is not notified of the reception of the pushed data. Most of the facilities provided by the connection oriented session service are confirmed , meaning that the client can send request primitives and receive confirm primitives and the server can send Response primitive and receive Indication primitives.

The connectionless session service provides only non confirmed services; in particular only unreliable method invocation (asking the server to execute an operation and return a result) and unconfirmed push are available. In this case clients can only use the Request primitive and servers are only able to use the indication primitive.

To start a new session the client invokes a WSP primitive that provides some parameters, such as the server address, the client address and the headers. These can be linked to the HTTP client headers and can for example be used by the server to retrieve the type of user agent within the wap client.

In some respects WSP is basically a binary form of HTTP. As previously mentioned the binary transmission of data between a server and the client is an essential adaptation made for the narrow bandwidth mobile network.

The WAP Session Protocol/B, WSP/B, is a stateless, binary protocol patterned after the HTTP World Wide Web protocol. It consists of a simple request-response pairing. WSP/B contains fields that describe the contents, origin, and types of the request or response contents. There is a one to one correspondence with a subset of the HTTP 1.1 fields. No state information is maintained between requests. WSP uses the WAP Datagram protocol directly for communication with WAP clients. WSP/B is sometimes called WAP connectionless mode.

The WAP Session Protocol, WSP, is a session oriented, stateful binary protocol used in conjunction with WTP. WSP is a superset of WSP/B and uses the same fields of information. WSP also defines additional protocol formats to support sessions initiation, suspension, and resumption and to maintain session state information. A session is initiated by a WAP client and is maintained until it is explicitly disconnected. WSP sessions can be suspended and resumed and can even switch WDP bearers mid-stream. All WSP information is exchanged using the WAP Transaction Protocol.

3.9. Transaction Layer

WTP (Wireless Transaction Protocol) provides different methods for performing transaction, to a varying degree of reliability. The Wireless Transaction Protocol (WTP) runs on top of a datagram service and provides as a light-weight transaction-oriented protocol that is suitable for implementation in “thin” clients (mobile stations). WTP operates efficiently over secure or non-secure wireless datagram networks and provides the following features:
Three classes of transaction service:

- Unreliable one-way requests,
- Reliable one-way requests, and
- Reliable two-way request-reply transactions;

WTP, as all the other layers in WAP, is optimized to adapt to the small bandwidth of the radio interface, trying to reduce the total amount of replayed transactions between the client and the server.

In particular, three different classes of transaction services are supplied to the upper layers:

- Unreliable requests
- Reliable requests
- Reliable requests with one result message

3.9.1. Unreliable requests

The initiator in this case a content server sends a request to the responder (the user agent) who does not reply with an acknowledgement. The transaction has no state and terminates once the invoked message is sent.

3.9.2. Reliable Requests

The Initiator sends a request to the responder (the user agent) who acknowledges it. The responder stores the transaction state information for some so that it can retransmit the acknowledgement message if the server request again.

3.9.3. Reliable Request with One Result Message

The initiator sends a request to the responder who implicitly acknowledges it with a result message, maintaining the transaction state information for some time after the acknowledgement has been sent, in case it fails to arrive .The transaction ends at the responder when it receives the acknowledgement message.

WTP is designed to run on top of a datagram protocol, e.g. WDP Datagrams are unreliable and to be able to provide a reliable service WTP uses techniques utilizing retransmission after timeouts and acknowledgement of messages. To be able to spot duplicate messages, WTP uses a transaction identifier (TID) in every message. The TID is used to associate a packet with a particular transaction. The TID is cached at the receiver and if a TID with an equal or lower number than the cached TID is received, the message MAY be a duplicate. It may also be a re-transmitted message which was lost earlier. The receiver is able to ask the sender if the TID is valid or not. WTP uses a primitive error handling. If an error occurs (eg. the connection is broken) the transaction is aborted.

3.10. Security Layer

WTLS (Wireless Transport Layer Security) is an optional layer that provides when present ,authentication , privacy and secure connections between applications.

WTLS is a security protocol 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 provides the following features:

- Data integrity – WTLS contains facilities to ensure that data sent between the terminal and an application server is unchanged and uncorrupted.

- Privacy – WTLS contains facilities to ensures that data transmitted between the terminal and an application server is private and cannot be understood by any intermediate parties that may have intercepted the datastream.

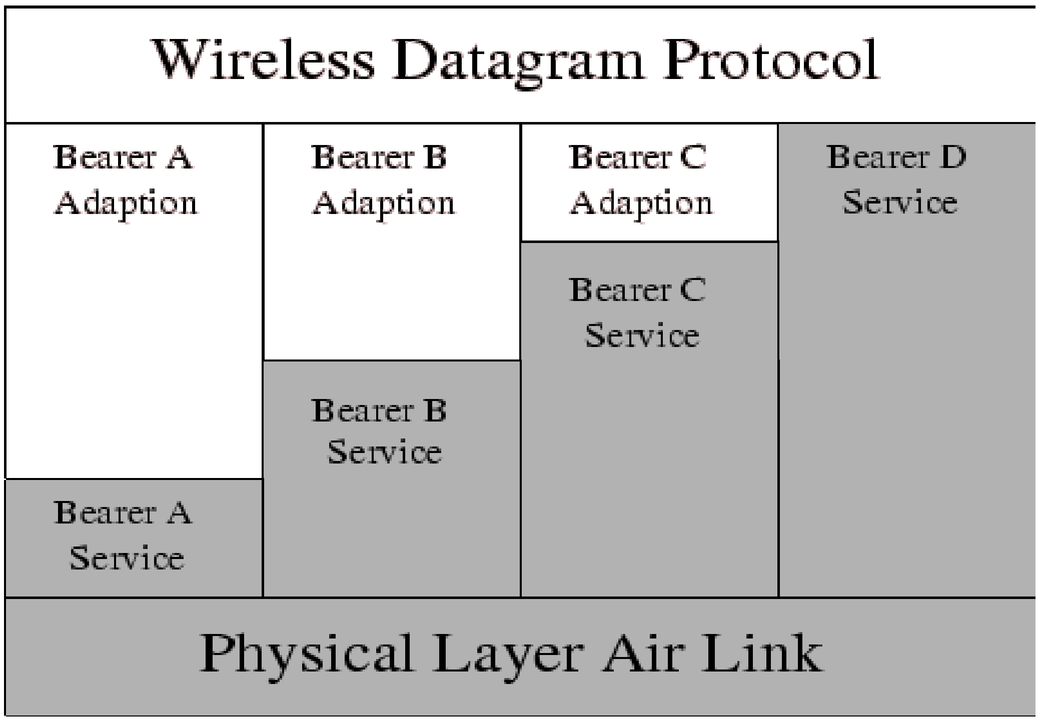
- Authentication – WTLS contains facilities to establish the authenticity of the terminal and application server.

- Denial-of-service protection – WTLS contains facilities for detecting and rejecting data that is replayed or not successfully verified. WTLS makes many typical denial-of-service attacks harder to accomplish and protects the upper protocol layers.

The WAP Transaction Layer Security, WTLS, is a session oriented, secure protocol layer patterned after the web's Secure Session Layer (SSL) and Transaction Layer Security (TLS) protocols. The WTLS layer is optional and is independent of the layers above and below it. One unique feature of WTLS is the ability of both client and server to independently recalculate encryption key information based on an embedded sequence number. WTLS is thus optimized to minimize information exchange between client and server. There are three levels of WTLS secure sessions. Level one is anonymous encryption where neither client nor server is authenticated. Level two supports server certificates where clients authenticate the server. Level three supports client certificates where the server can authenticate the client. WTLS supports three certificate types: x.509, WTLS, and x.968. The WTLS certificate format is unique to WAP and is designed to minimize information transfer. The x.509 certificate is the same format as that used on the web in SSL and TLS transactions. And the x.968 format is currently not fully specified, but will be supported in the future. WTLS is compatible with both WSP/B and WSP with WTP and can is activated as an additional protocol layer between either of these higher layers and the WDP protocol.

3.11. Transport Layer:

WDP (Wireless Datagram Protocol) is the bottom layer of the Wap stack, which shelters the upper layers from the bearer services offered by the operator. The WAP Datagram Protocol, WDP, is a datagram oriented, network layer protocol modeled after the User Datagram Protocol (UDP) used on the Internet. UDP is a member of the TCP/IP protocol suite and is a simple, "best effort" data delivery protocol. On those networks where Internet protocols are present, WDP and UDP are identical. On networks where UDP is not available, WAP defines a UDP equivalent. These UDP equivalents are known as "mappings". The currently defined mappings create the equivalent of UDP over SMS, USSD, and other mobile data transports. WDP makes no attempt to confirm delivery, resend lost packets, or correct errors in transmission. This is left to the higher layer protocols.



Rather than specifying a protocol, Wireless Datagram Protocol (WDP) specifies how different existing bearer services should be used to provide a consistent service to the upper layers. This is done by adapting the protocol to the underlying bearer as shown. For e.g. for an IP bearer the WDP specs simply states that as WDP you must use the UDP protocol from the IP-suite.

The varying heights of each of the bearer services shown in above figure illustrates the difference in functions provided by the bearers and thus the difference in WDP protocol necessary to operate over those bearers to maintain the same service offering at the Transport Service Access Point is accomplished by a bearer adaptation.

WDP can be mapped onto different bearers, with different characteristics. In order to optimize the protocol with respect to memory usage and radio transmission efficiency, the protocol performance over each bearer may vary. However, the WDP service and service primitives will remain the same, providing a consistent interface to the higher layers.

3.12. Bearers

The WAP protocols are designed to operate over a variety of different bearer services, including short message, circuit-switched data, and packet data. The bearers offer differing levels of quality of service with respect to throughput, error rate, and delays. The WAP protocols are designed to compensate for or tolerate these varying levels of service.

Since the WDP layer provides the convergence between the bearer service and the rest of the WAP stack, the WDP specification [WDP] lists the bearers that are supported and the techniques used to allow WAP protocols to run over each bearer. The list of supported bearers will change over time with new bearers being added as the wireless market evolves.

3.13. Services and Applications

The WAP layered architecture enables other services and applications to utilize the features of the WAP stack through a set of well-defined interfaces. External applications may access the session, transaction, security and transport layers directly. The WAP layered architecture enables other services and applications to utilize the features of the WAP stack through a set of well-defined interfaces. External applications may access the session, transaction, security and transport layers directly. This allows the WAP stack to be used for applications and services not currently specified by WAP, but deemed to be valuable for the wireless market. For example, applications, such as electronic mail, calendar, phone book, notepad, and electronic commerce, or services, such as white and yellow pages, may be developed to use the WAP protocols.

Chapter 4
WAP Gateway

4.1. What is a WAP Gateway?

A WAP Gateway forms a bridge between two distinct worlds the internet (or another IP packet network) and the wireless phone/data network, which are fundamentally different in their underlying technologies.

Work is currently being done into the convergence of various technologies that will make life simpler for people who access information .Eventually we may see a day when a single predominant technology will be used for all types of network, supporting voice, data and video services .However until then we need solution specific technologies, like WAP to enable information flow towards users who are using different access mechanisms.

A wap gateway is basically software that is placed between a network that supports wap and an IP packet network such as the Internet. It acts as a intermediary that converts between the protocols of the packet network and the protocols of the wap network (WSP, WTP, WTLS and WDP). When cellular packet networks, such as GPRS that can use TCP/IP directly are prevalent, it may make more sense to use the WAP protocols as its nature is to reduce the data transfer sizes required. In addition WAP for the moment presumes the use of wml which is geared towards small screens and low processing power. If on the other hand you use a GPRS mobile connected to a laptop we can access HTTP and TCP/IP directly to access information on the internet. Among other things the gateway converts WSP requests from wireless devices into HTTP requests and vice versa for the HTTP responses.

A WAP gateway can be implemented as a single host or a cluster of servers for load balancing. However regardless of the implementation it can still be considered as a single box from a mobile user's perspective.
The internet is based on the TCP /IP protocol stack, which is suited to wired networks and quite unsuitable for most types of wireless networks. This is because
TCP is heavyweight transport layer that has high overhead especially during connection establishment. This is due to the three way handshake mechanism. it also transmits large amounts of data to handle the possibility of packets arriving in a

different order to which they ere sent. This could happen if the packets take different routes in an IP network.

By keeping the Wap communication protocol stack separate from that of the internet, it is possible to implement WAP for a wide range of wireless bearer networks (internet protocols are not suitable for such types of networks). This insures bearer's independence for all layers of the stack except WDP. Because WDP has to interface to the upper protocol layers , so it must have a bearer specific implementation . If a vendor develops the wap protocol stack , WDP is the only layer that must be re written to support different bearer networks . The WTP layer implements a simple request-response transaction oriented protocol instead of the three way handshake connection – establishment mechanism. WTP has been inspired by TCP.

4.2. Functionality of a WAP Gateway

Many of the functions of a wap gateway are optional. Below is a summary of functions that a gateway carries out.

- Implementation of wap protocol stack layers
- Access Control
- Protocol conversion
- Domain Name resolution
- HTML to WML conversion
- Encoding of WML content
- WMLScipt Compilation
- Security
- Caching

As the goal of our project was to create Gateway with minimal functionality work on the following from above were carried out.

- Implementation of Wap protocol stack layers
- Protocol conversion
- Encoding of wml content

4.3. Implementation of WAP protocol stack layers

This is the most obvious function of a wap gateway and it contributes to most of the functionality of a gateway. Depending whether the type of service is connection oriented or connectionless secure or non-secure the following stack layers need to be implemented:

• Non –secure Connection oriented	WSP	WTP		WDP
• Secure connection oriented	WSP	WTP	WTLS	WDP
• Non –secure Connectionless	WSP			WDP
• Secure Connectionless	WSP		WTLS	WDP

We have implemented the Non-secure Connectionless service.

4.4. Protocol conversion: WSP ⇔ HTTP

WSP supports complete HTTP functionality. This includes request reply methods (like GET, POST), request response and entity headers (like “Accept application /vnd.wap.wml”) a request header that specifies the particular MIME types that a client can handle) and content negotiation .Content negotiation is the process of selecting the best representation suited for a client for a given response when there are multiple representations for the same content available from the server.

A request Header is a meta information that is sent along   with a HTTP request (like GET or POST requests).Similarly a response header is meta information in a http response that is sent by the server as a response to a previous HTTP request
As part of the response the server might also send in a entity body depending on the type of the request. The meta information sent to give more meaning to the entity body that was sent is known as an entity header.

However, WSP headers are in compact binary tokenized format as defined in the WSP specification. A token is a group of characters that has a specific meaning when used together as a string. For example in the Accept header are all string tokens. A binary token for these would be an octet representation.

Accept: text/plain, text/vnd.wap.wml, text /vnd.wap.wmlscript, application/vnd.wap.wmlc,application /vnd.wap.wmlscriptc

The above request header indicates to the server that the client can accept content in any of the above MIME formats, plain text, wml and wmlscipt in plain and compact form.

Using WSP, the same header is represented as
0x80 0x83 0x88 0x94 0x95

4.5. Comparison between HTTP and WSP

The Wireless Session Protocol is a bigger protocol than the Hypertext Transfer Protocol. Bigger in the sense that WSP supports more functionality then HTTP does, e.g. pushing content from the server to the client and multiple simultaneous asynchronous transactions. This chapter will describe the Hypertext Transfer Protocol and the method invocation facility in the Wireless Session Protocol.

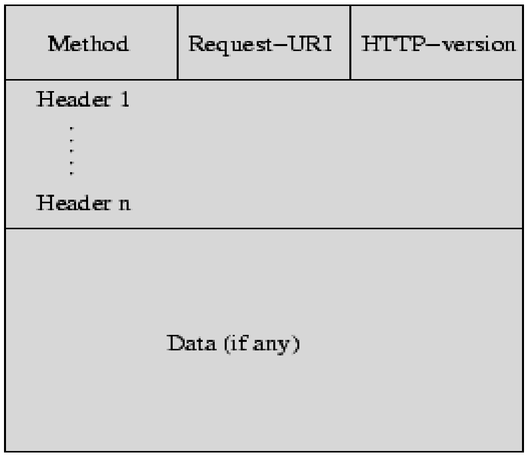
4.5.1. Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is a protocol used by web clients and web servers. It's defined by the document *Hypertext Transfer Protocol - HTTP/1.1* . The protocol is a request-response protocol, which means that the client first makes a request and then the server answers with a response. A method is supplied in the request and one can say that the client invokes a method on the server. Each request or response is called a message and consists of these parts:

1. First line
2. Headers
3. Data

The first line determines if the message is a request or a response. The headers that follow give information about the request/response and also about the data. A header is one line and it ends with a CRLF sequence. The data part is the information which is to be sent between the client and server. This part may be empty if there is no information to be sent.

4.5.1.1. Request

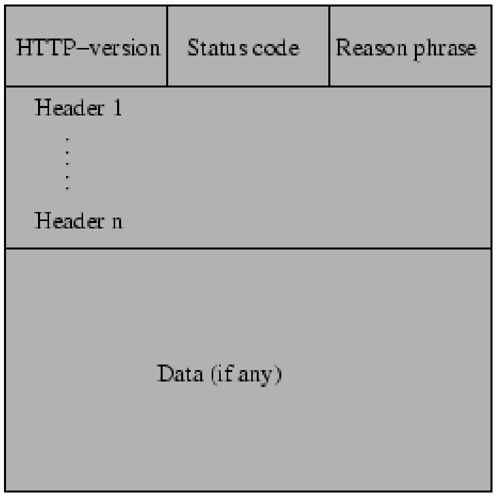


In a request the first line states which method the client wishes to invoke on the server. The methods defined in HTTP/1.1 are *options*, *get*, *head*, *post*, *put*, *delete*,

trace and *connect*. The first line also contains the URI to a document and the HTTP version the client is using.
The headers in a request gives the server additional information about it. They could for instance instruct the server not to send a web page if it hasn't been modified since a particular date or they could notify the server of which languages the client (or the user) understands. The data part of the message may be omitted if there is no information to be sent to the server. The only methods that actually have a data part are POST and PUT. They are used to send information or upload files to the server.

4.5.1.2. Response

The first line of a response states which version of HTTP the web server is using. It also contains a status code stating either the request has been fulfilled or what type of error the server encountered while trying to fulfill the request. The status code comes with a small *reason phrase* which describes the status code.



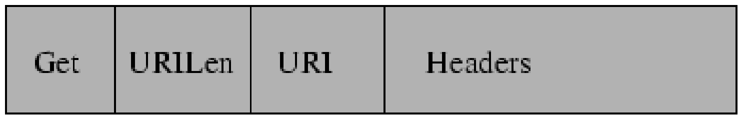
The response also includes headers. These headers contain important information about the data within the message, for instance a header describing how the content should be interpreted (e.g. if the contents should be interpreted as HTML text, a GIF image or some other data). Almost all responses contain some data intended for the client. If there is an error, most web servers send a HTML page describing this error and most web browsers display this page to the user.

4.5.2. WSP Method Invocation

The method invocation facility in the WSP protocol is designed to give the same functionality as the HTTP protocol, but with a shorter syntax. Headers in HTTP are ASCII lines ending with CRLF. The WSP protocol does not have a special mark to distinguish between headers. Instead, the specification assigns a byte number to each header present in HTTP. The specification also assigns small byte sequences for the values or rules to encode the values of the HTTP headers. If a HTTP header is not to be found in the WSP specification, the header is considered to be an application header and is sent in plain text.

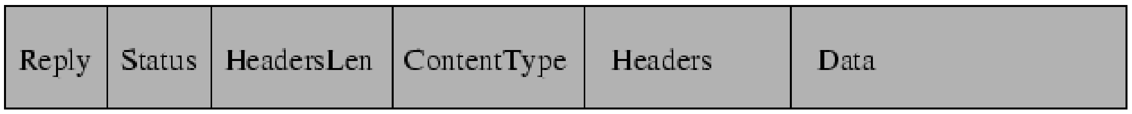
4.5.2.1. Request

WSP uses a PDU with the code for *Get* in the type field to make a request. Following the type, there is an integer, URILen, stating how long the URI is and after the URILen, the URI itself. The headers in a *Get* consist of encoded HTTP headers and application specific headers and reach until the end of the PDU. Figure shows a picture of a *Get* PDU (omitting the transaction id at the beginning).



4.5.2.2. Reply

When the server has gotten a request, it will answer with a reply. The reply contains the same information as a HTTP response. Right after the type field, there is a Status code byte, which is an encoding of the HTTP status code. The reason phrase is not sent in a WSP reply since it is redundant. An integer, HeadersLen, states how much space (in bytes) the encoded headers will occupy in the PDU. The fist header in the reply must be the ContentType header. This header tells the client how to interpret the data. After the ContentType header, all other headers follow. The headers are followed by the data begins which reaches until the end of the PDU. Figure shows a picture of a *Reply* PDU (omitting the transaction id).



4.6. Encoding of WML Content

WML content coming from the internet or another provider –is encoded into a compact binary form at the gateway before it is sent to the wireless device. This process is known as tokenization. During the process the gateway also performs checks to verify that the wml content has no errors and is well formed .(Because WML is an XML language , it has to comply with the well formedness of XML , the rules set out to define correctly formed XML.. In this case where the verification fails the gateway sends an error indication to the user agent on the wireless device.

Chapter 5
JAVA as a Language

5.1 What Is Java?

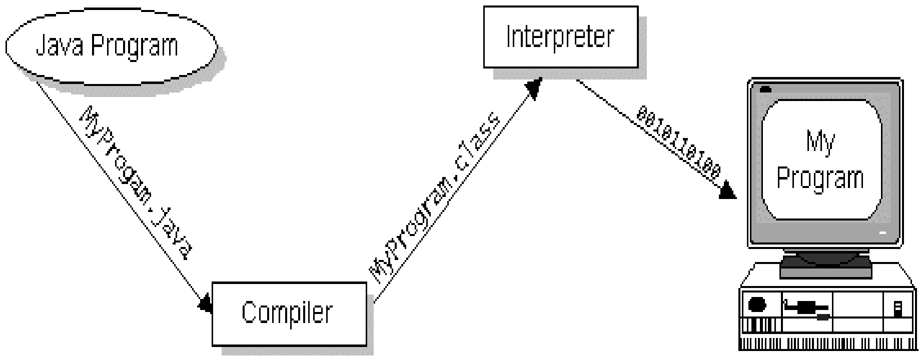
Java can be thought of composed of two things: a programming language and a platform.

5.2 Java as Programming Language

Java is a high-level programming language that is all of the following:

Simple	Architecture-neutral
Object-oriented	Portable
Distributed	High-performance
Interpreted	Multithreaded
Robust	Dynamic
Secure	

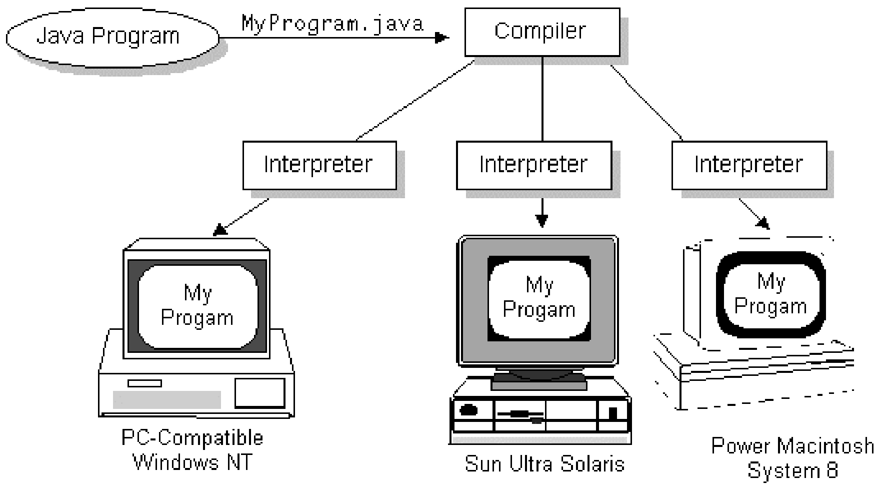
Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called *Java bytecodes*--the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed.



This figure illustrates how this works.

You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java bytecodes help make "write once, run anywhere" possible. You can compile your Java program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT,
Solaris, and Macintosh.



5.3 Java as Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

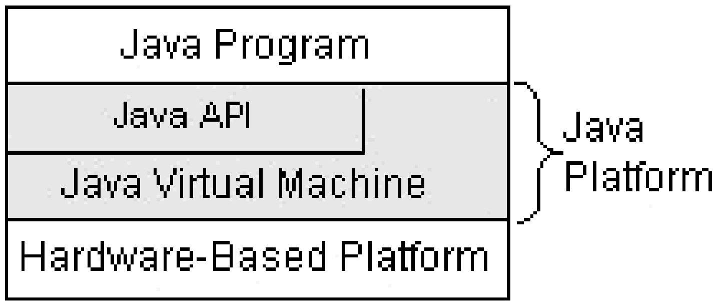
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

JVM stands for Java virtual machine .It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (*packages*) of related components. The next section, highlights each area of functionality provided by the packages in the Java API.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.



As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java's performance close to that of native code without threatening portability.

5.4 What Can Java Do?

Probably the most well-known Java programs are *Java applets*. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser. At the beginning of this trail is an applet that displays an animation of Java's mascot, Duke, waving at you.

However, Java is not just for writing cute, entertaining applets for the World Wide Web ("Web"). Java is a general-purpose, high-level programming language and a powerful software platform. Using the generous Java API, you can write many types of programs.

The most common types of programs are probably applets and applications, where a Java application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers include Web servers, proxy servers, mail servers, print servers, and boot servers. Another specialized program is a *servlet*. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java servers, configuring or tailoring the server.

How does the Java API support all of these kinds of programs? With packages of software components that provide a wide range of functionality. The *core API* is the API included in every full implementation of the Java platform. The core API gives you the following features:

- **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by Java applets.
- **Networking**: URLs, TCP and UDP sockets, and IP addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.

- **Software components**: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.

- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

- **Java Database Connectivity (JDBC)**: Provides uniform access to a wide range of relational databases.

  Java not only has a core API, but also standard extensions. The standard extensions define APIs for 3D, servers, collaboration, telephony, speech, animation, and more.

5.5 Advantages of Java

We can't promise you fame, fortune, or even a job if you learn Java. Still, Java is likely to make your programs better and requires less effort than other languages. We believe that Java will help you do the following:

- **Get started quickly**: Although Java is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.

- **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in Java can be four times smaller than the same program in C++.

- **Write better code**: The Java language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Java's object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.

  - **Develop programs faster**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code with Java and Java is a simpler programming language than C++.

- **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by following the purity tips mentioned throughout this book and avoiding the use of libraries written in other languages.
- **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the Java feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

Chapter 6
Structured Analysis

6.1 ENVIRONMENTAL MODEL:

The environmental model defines the boundary between the system and the rest of the world and consists of 3 different activities:

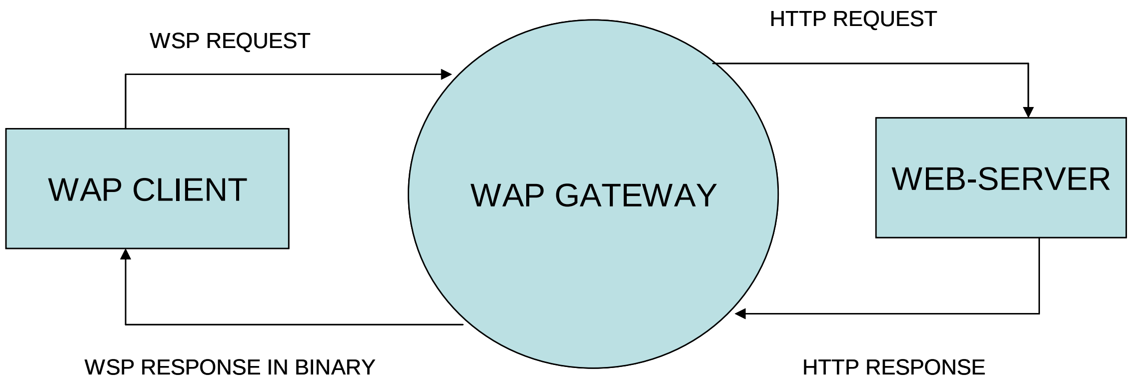
- A Statement of Purpose
- A Context Diagram
- An Event List

6.1.1. Statement of purpose:

A WAP Gateway is an implementation of wap stack which translates between protocols in the WAP protocol suite and the TCP/IP protocol suite. The wap gateway we are going to implement will perform two basic functions:

- It will act as a translator between Wireless Session Protocol (WSP) and Hypertext Transfer Protocol (HTTP).
- The WAP gateway will also be responsible for the conversion of WML content into binary format.

6.1.2. Context Diagram:



6.1.3. Event List

An event list is the list of all the events that are expected out of the system. The event list for the present system can be stated as follows:

- Phone user enters a URL to a specific document.
- Phone sends a WSP request to the gateway.
- Gateway parses the request.
- Gateway sends a HTTP request for the document specified in the URL.
- HTTP server answers with a response to the gateway.
- Gateway parses the response.
- If the content type is WML then the gateway compiles the data into binary WML.
- Gateway sends a WSP reply to the mobile phone.
- Mobile phone presents the document to the user.

6.2. Behavioral Model:

The behavioral model describes the required behavior of the insides of the system necessary to interact successfully with the environment. The behavioral modeling can be explained using the Data Flow Diagrams.

6.2.1. Data Flow Diagrams:

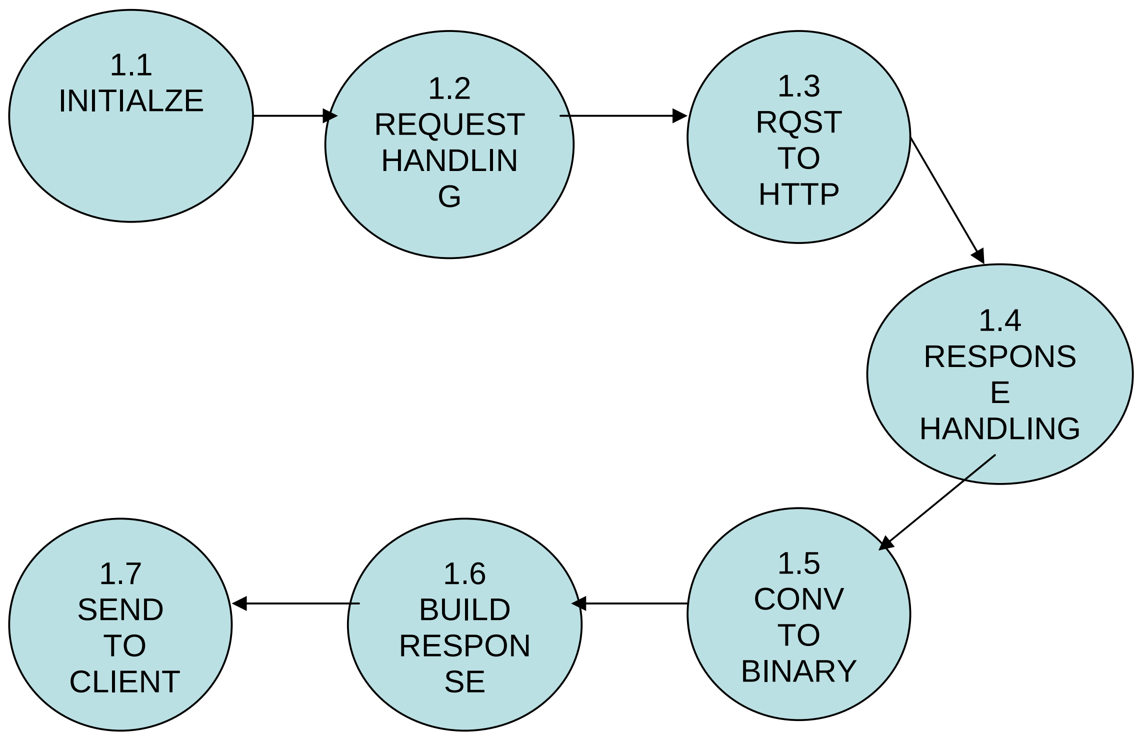
A data flow diagram (DFD) is a graphical technique that describes information flow and the transformations that are applied as data move from input to output. It is a graphic tool and can be used for system analysis by focusing on the functional aspect of the system to be developed. That is, it helps answer the two questions

"What will the functions of the system accomplish?" and "What are the interactions between those functions?".

The multiple level DFD's for the system are as follows:

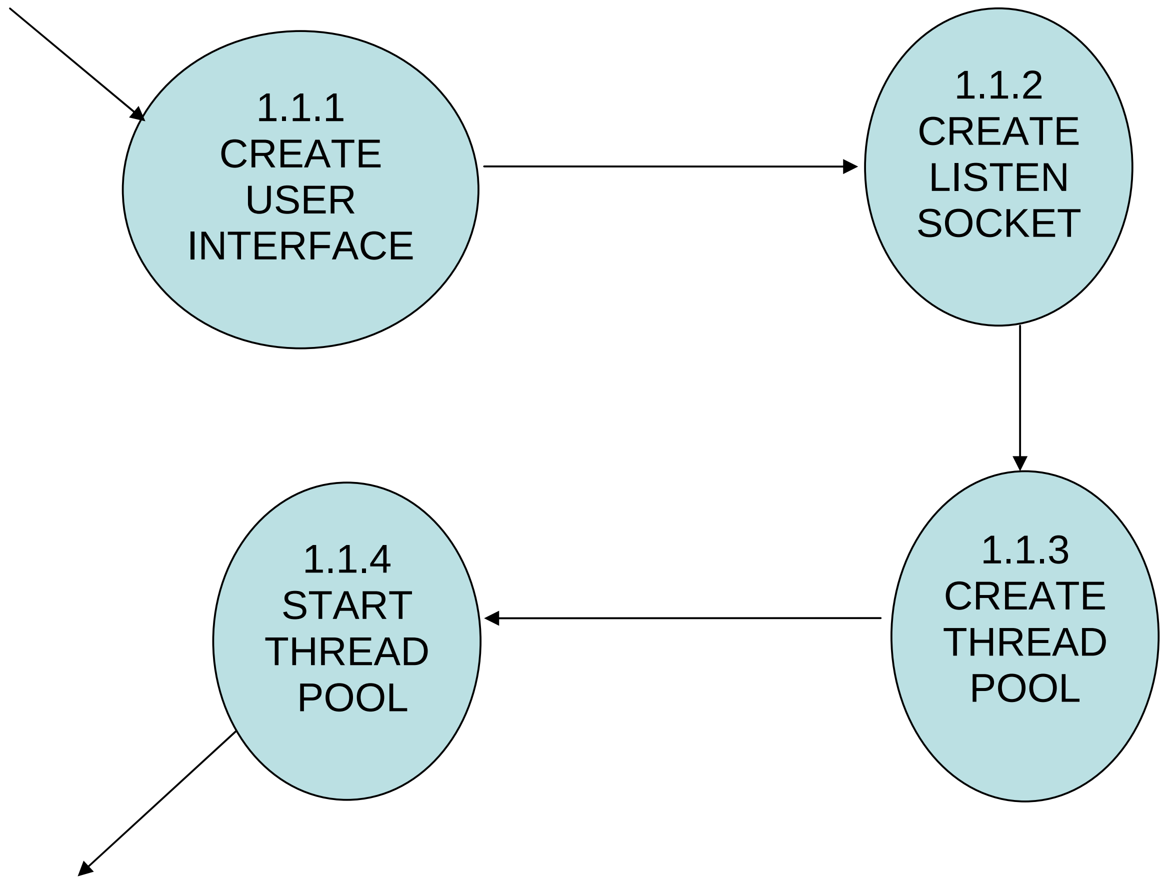
6.2.1.1. Level 1 DFD:

The level 1 DFD for the system is as follows:



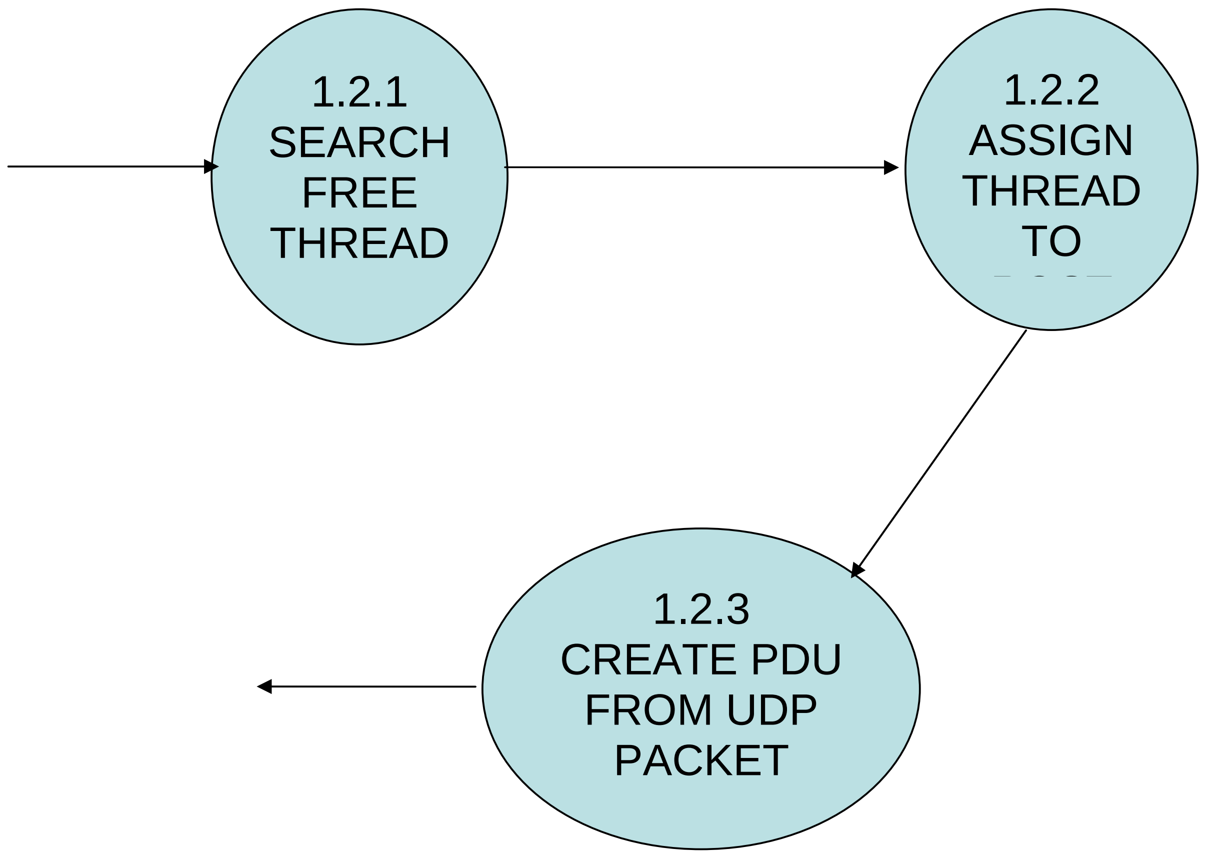
6.2.1.2. Level 2 DFD (Bubbling of 6.1.1):

The bubbling of the first bubble i.e. (1.1) gives the following result.



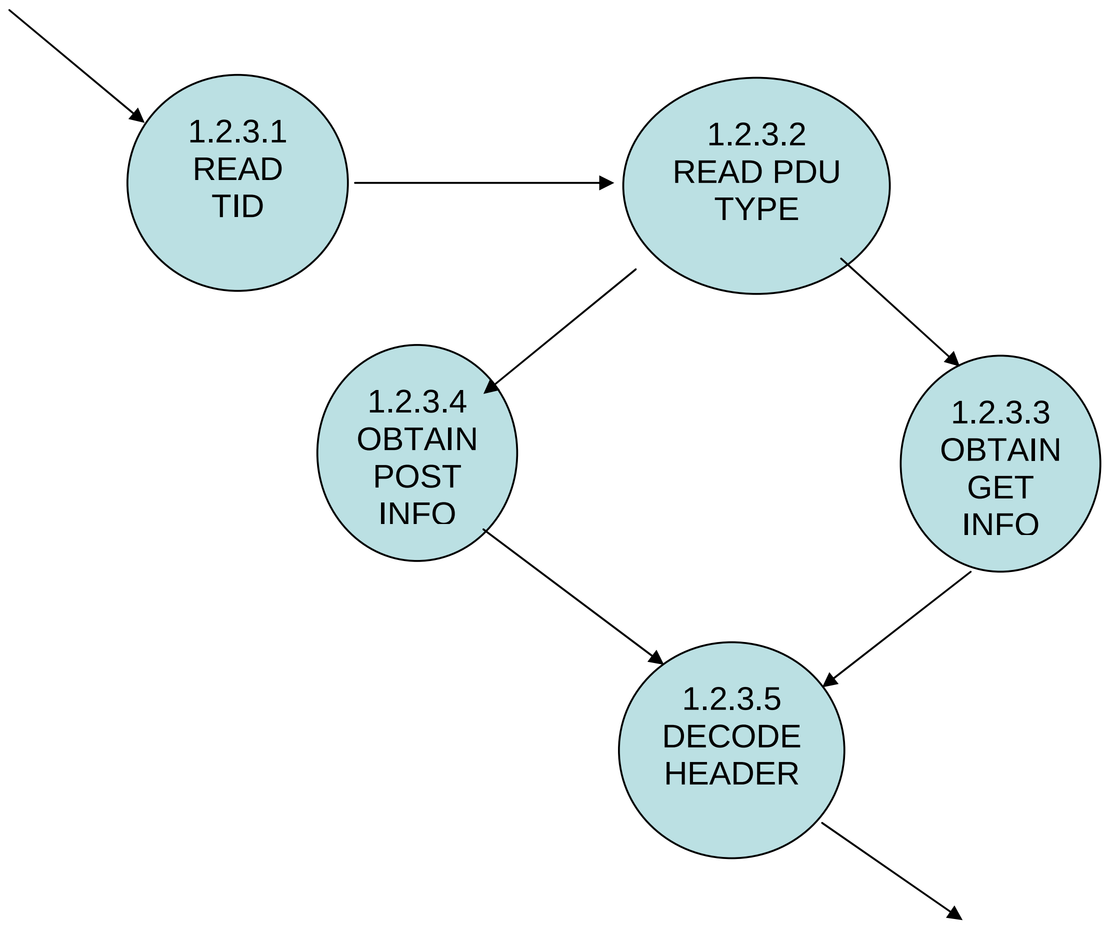
6.2.1.3. Level 2 DFD (Bubbling of 6.1.2):

The bubbling of bubble 6.1.2 gives the following DFD at level 2.



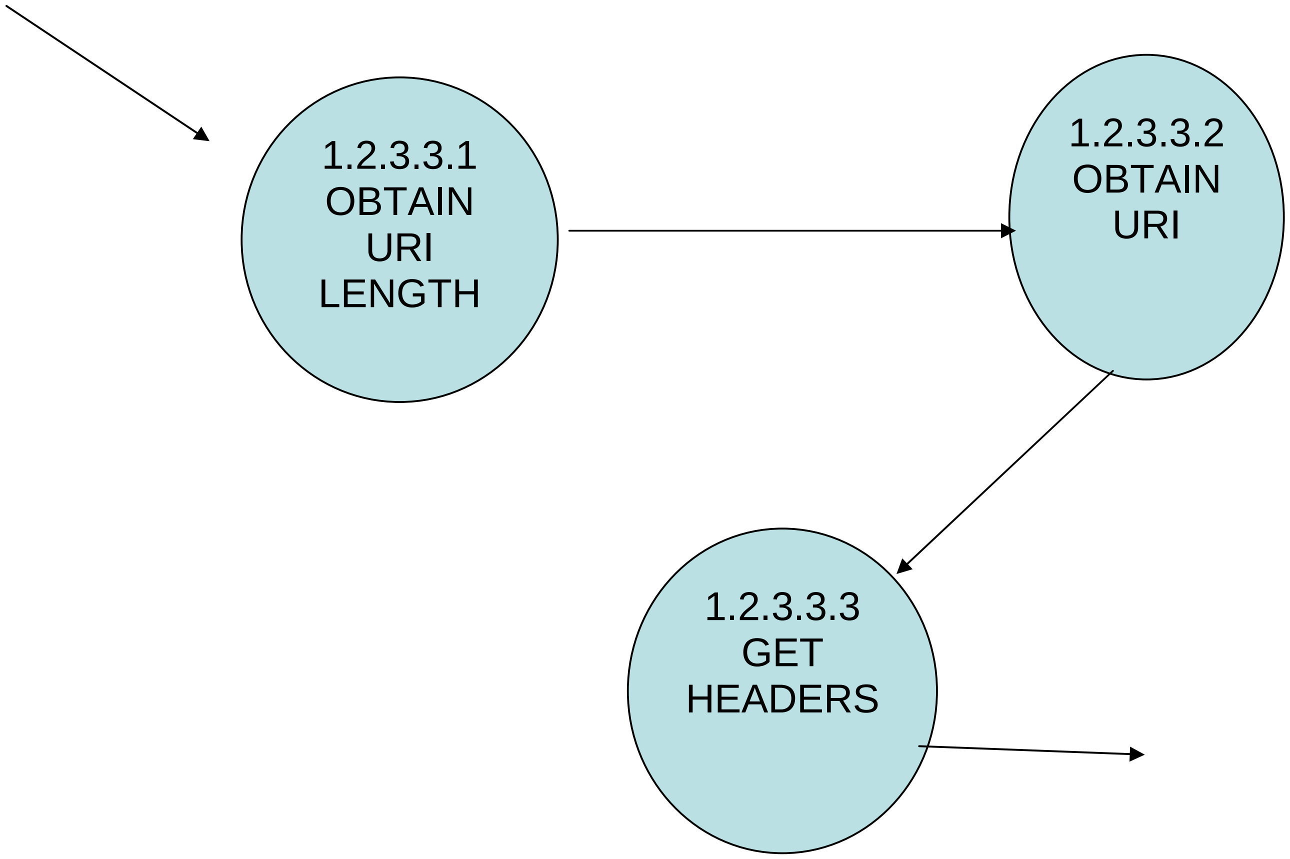
6.2.1.4. Level 3 DFD (Bubbling of 1.2.3):

The further bubbling of process 1.2.3 gives the following result:



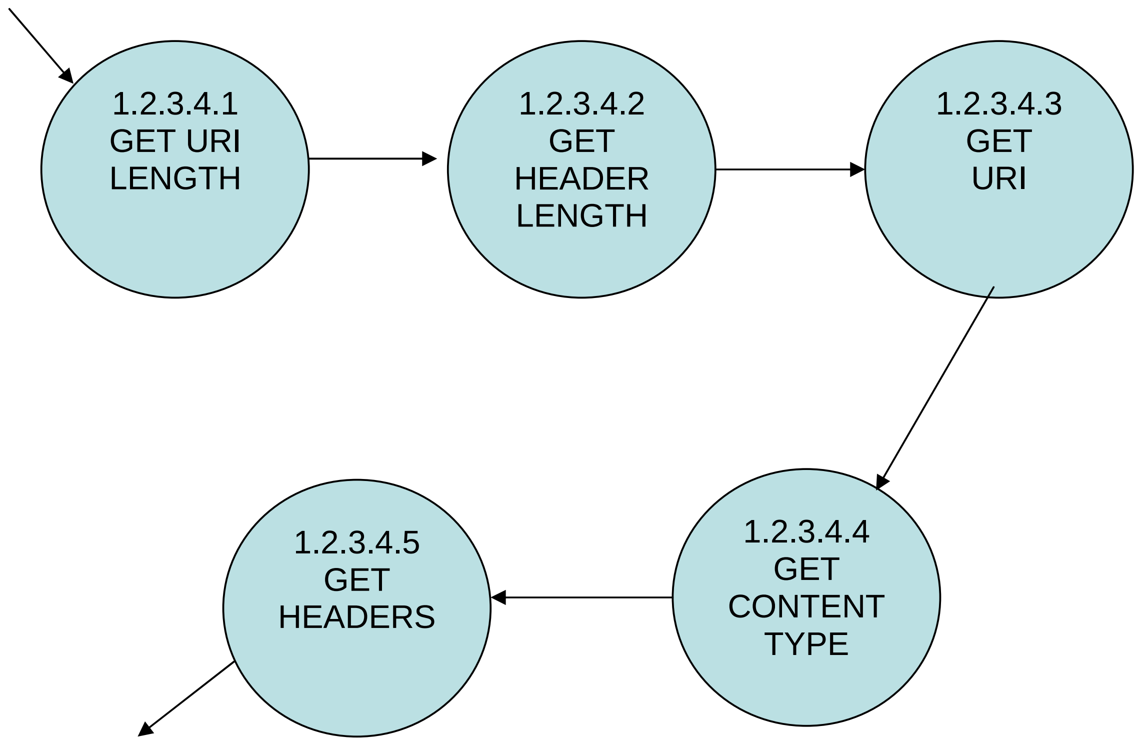
6.2.1.5. Level 4 DFD (Bubbling of 6.1.2.3.3):

Further subdivision of the level 3 bubble i.e. 1.1.3 gives the following:



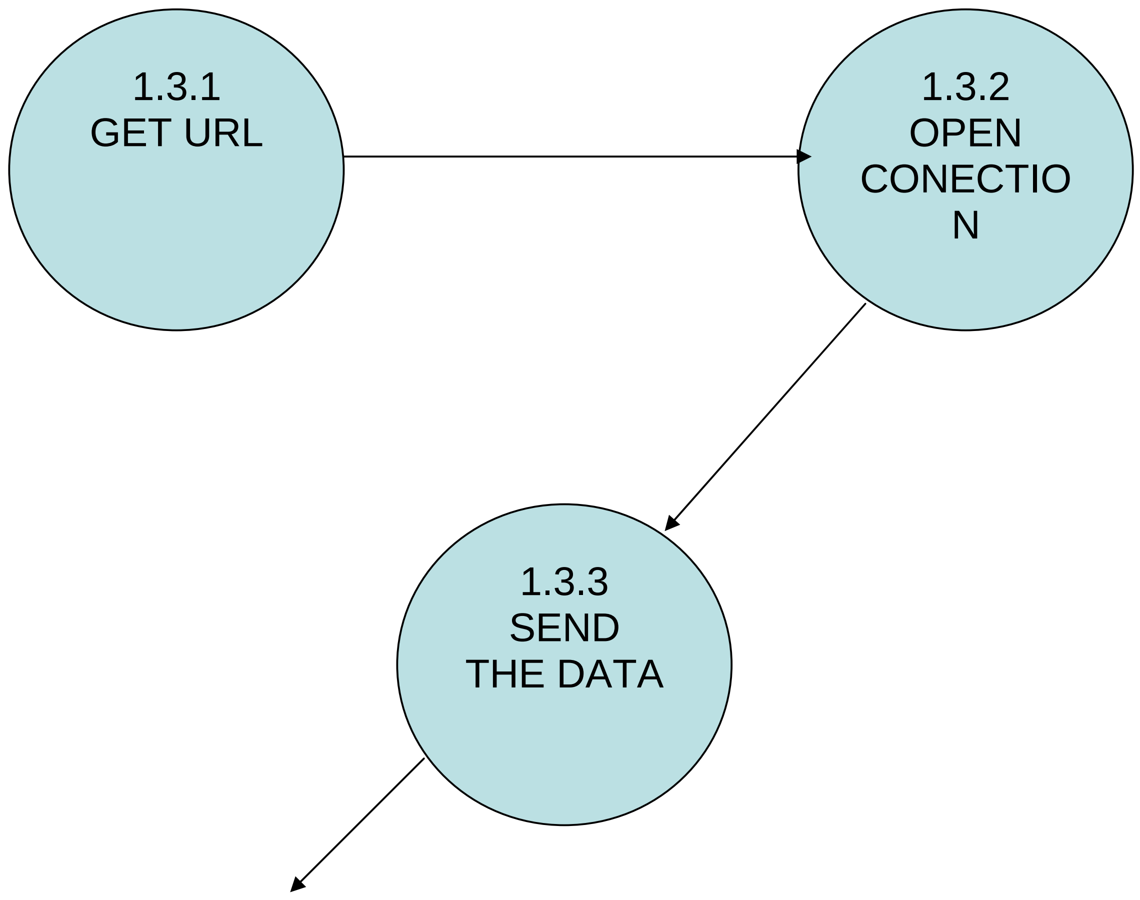
6.2.1.6. Level 4 DFD (Bubbling of 6.1.2.3.4) :

Further bubbling of level 3 process 1.2.3.4 gives:



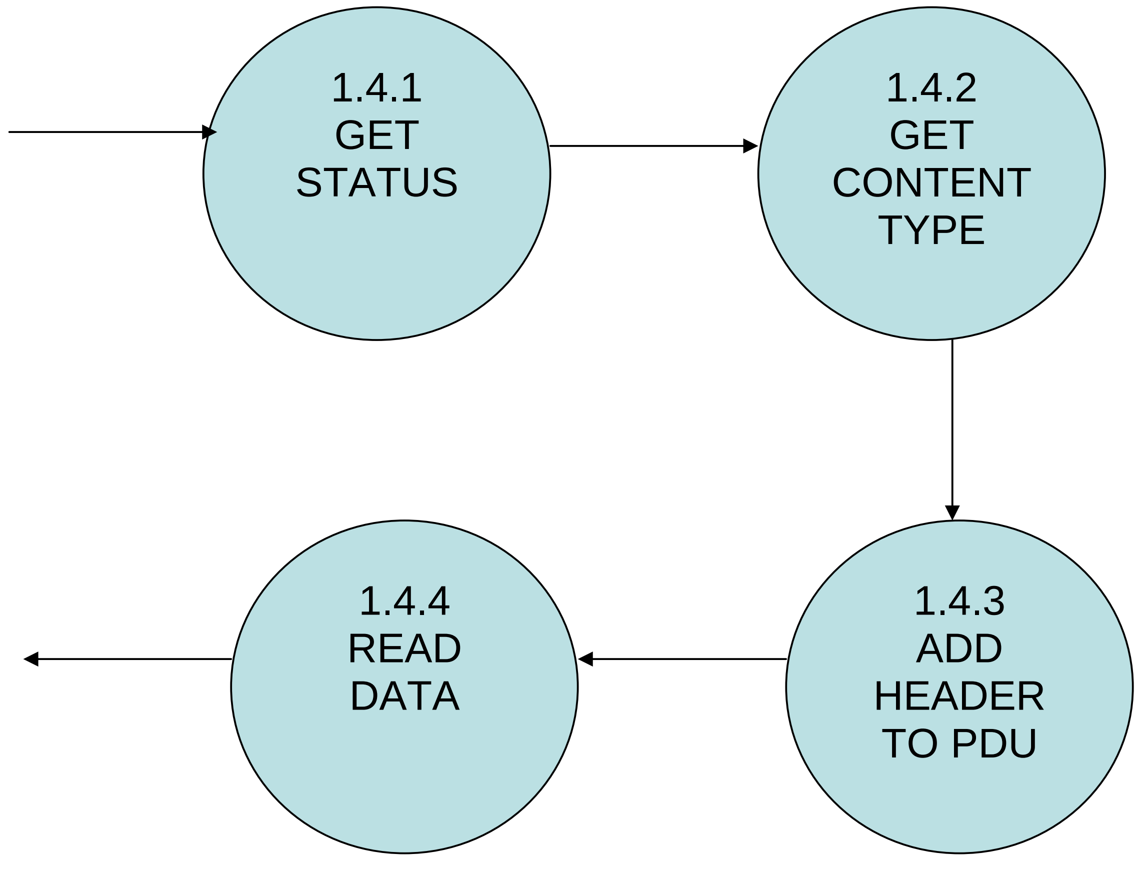
6.2.1.7. Level 2 DFD (Bubbling of 6.1.3):

The bubbling of level 2 process 1.3 gives further DFD as follows:



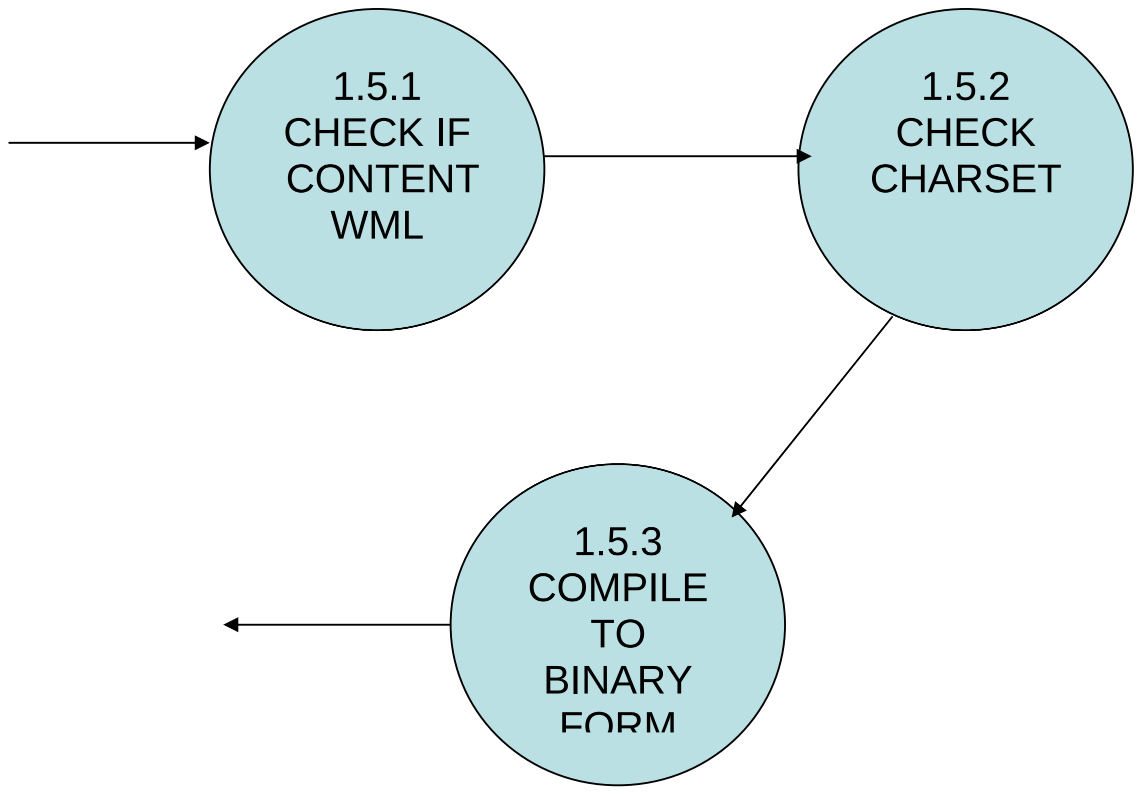
6.2.1.8. Level 2 DFD (Bubbling of 6.1.4):

The bubbling of the level 1 DFD gives the further DFD as follows:



6.2.1.9. Level 2 DFD (Bubbling of 6.1.5):

The further bubbling of level 1 process 1.5 gives the following DFD:



Chapter 7
Software Design

7.1 Class gateway:

This class is the main class of the project. This class has got various responsibilities which include the initial display of the information. Creation of the user interface. Building the datagram socket to listen for the requests. So all the preliminaries of the gateway are handled by this class.

Class Gateway

Button b;
Process p;
Runtime r;
DatagramSocket sock;
TextArea txa;

Gateway ();
Public static void main (String a []);
Public void send (DatagramPacket);
Public void actionPerformed
(ActionEvent);

7.1.1. DatagramSocket:

This variable is used to create a datagram socket so that the data can be received and send on that socket.

7.1.2. Gateway ():

This method is the constructor of the class. As this class is called from the main class of the program so as soon as the program is run, this method create the user interface.

7.1.3. main (String args[]):

This method is the main method of the whole program. The program starts from here. It is responsible for creating DatagramSocket. It also sets up a pool of threads. As soon as the request is received from the socket it fires up a new thread to handle the request.

7.1.4. actionPerformed (ActionEvent e):

This method is there because the class extends the ActionListener. This method is used for various controls on the user interface screen.

7.2 Class RequestHandler:

This class is basically there to serve a single request made by the WAP client. It creates a PDU from the UDP packet. Sends the request to http server. Gets the response from the http server and then sends the response back to the WAP client.

Class RequestHandler

Boolean free;
DatagramPacket p;

Public synchronized void run(); Public synchronized void setPacket(DatagramPacket);

7.2.1. free:

This variable is there for the indication of whether the present thread is free or is already handling a request. Once a thread is handling a request this variable is set so as to indicate that a request is already being handled.

7.2.2. run():

As the request class extends thread so the implementation of run method is a must. This method is of fundamental importance to all the request handling because it invokes various other methods to handle the requests.

7.2.3. setPacket():

This method sets the packet by copying the incoming packet into its own variable ‘p’ so that the request can be handled and also starts up the thread.

7.3 Class Http:

This class describes an Http Header with the field name and an array of field values. The basic function of this class is to encode an http header into a WAP encoded header. This class has two inner classes as well. These two classes are:

a. private class FieldValue
b. private class StrangeHTTPException

The FieldValue class is there for used by Http class to store field values and parameters. The other class is used as an exception class, if a code is not present in the table.

7.3.1. appendToFieldValue(String):

This method appends the extra field value to the field values.

7.3.2. String fieldName():

This method returns the HTTP field name.

```
Class Http
----------------------------------------
private String fName;
private String fValue;
----------------------------------------
void appendToFieldValue(String);
String fieldName();
String fieldValue();
HTTPHeader(String);
HTTPHeader(String, String );
private ByteVector makeCacheControlValue(String);
private ByteVector makeChallenge(String);
private ByteVector makeCredentials(String);
private byte[] makeqValueBytes(double);
private byte[] qValue12(double);
private byte[] qValue3(double);
void setValue(String);
private HTTPHeader[] splitListHeader();
private String[] splitString(String str,char separator);
private String[] splitValues(char separator);
private byte[] toExtensionMedia(String str);
private byte[] toQuotedString(String str);
ByteVector toWSP();
```

7.3.3. String fieldValue():

This method returns the field value at specified index.

7.3.4. HTTPHeader(String):

Constructs and fills in the name and values from a HTTP header. The parameter String is the HTTP Header to be processed.

7.3.5. HTTPHeader(String, String):

Constructs a HTTP header with the given field name and field value. The first parameter being the field name and the second being the field value.

7.3.6. makeCacheControlValue(String):

The cache control value is one of the header fields and this method is used to encode that value from the http header.

7.3.7. makeChallenge(String):

Makes a challenge from a header field value. It is also one of the fields in the http header. This field is used for authorization schemes.

7.3.8. makeCredentials(String):

Credentials is also one of the fields used for authorization or authentication schemes. This method uses the given field name for making credentials.

7.3.9. makeqValueBytes(double):

This method returns byte array ready to be sent. It returns null if qValue is 1.

7.3.10. qValue12(double):

This method is to be used when encoding q-values with 0, 1 or 2 decimal digits. It returns an array of bytes containing the codes.

7.3.11. qValue3(double):

The method is used when encoding q-values with 3 decimal digits. It returns an array of bytes containing the codes.

7.3.12. setValue(String):

This method sets the field value to the given string value.

7.3.13. splitListHeader():

This method splits a comma separated list of header field values into mutiple headers.

7.3.14. splitString(String ,char):

Splits a string into substrings using separator to separate them.

7.3.15. toWSP():

Encodes a HTTP Header into WAP encoded header and returns a ByteVector containing the bytes making up the header.

7.4. Class ThreadPool:

This class is used to hold a pool of threads ready to take care of the incoming requests. The class keeps of the number of the unused threads at an appropriate level. The class is responsible for firing of a new thread when a new request comes in.

Class ThreadPool

private Vector threads;
private long interval;

ThreadPool(int, long);
void dispatch(DatagramPacket);
public void run();

7.4.1. ThreadPool(int, long):

Creates a new pool with initialNo of threads indicated.

7.4.2. dispatch(DatagramPacket):

This method dispatches a thread to serve the incoming Datagram packet.

7.4.3. run():

As the class extends the Thread class so the implementation of run() method is a must. The run method is used to keep track of unused threads and also searches for the free threads once a request comes in.

7.5. Class HttpDecoder:

This class decodes a vector containing bytes into one HTTPHeader. The request coming from the WAP client contains a http header but it is encoded in the request. So this class is used to decode that vector.

Class HttpDecoder

private final Vector CTL=new Vector(33);
private final Vector CRLF=new Vector(2);
private final Vector CHAR=new Vector(127);
private final Vector separators = new Vector(19);

public Object clone();
HTTPHeader decodeHeader();
public PDU mkPDU();
public int read();
public HttpDecoder(InputStream in);

7.5.1. clone():

This method clones the stream in a certain sense. What it does is produce a new stream containing all the bytes that this clone contained at the moment.

7.5.2. decodeHeader():

This method creates a HTTPHeader with the contents of the stream.

7.5.3. mkPDU():

This method pops a PDU from the Input stream.

7.5.4. read():

This method overrides read() but throws IOException if there's nothing to be read.

7.5.5. HttpDecoder(InputStream):

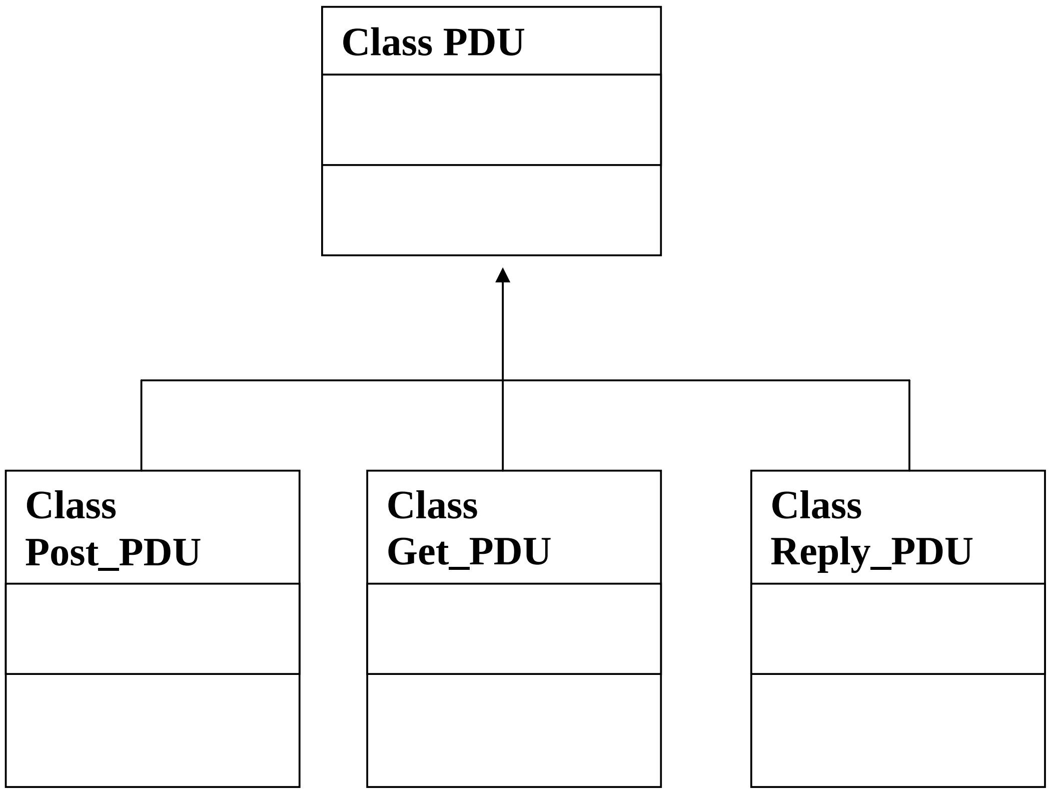
Creates a WAP Input Stream on the InputStream. This method also initializes the constants CTL, CRLF, CHAR and separators according to the RFC 2616 specifications.

7.6. Class PDU:

This class is responsible to hold a PDU no matter of what kind. There are basically there types of PDU's as can be thought of:

1. Get_PDU.
2. Post_PDU.
3. Reply_PDU.

Accordingly three subclasses have been defined in the class PDU. So the PDU class diagram can be shown as follows:



7.5.1 Class Get_PDU:

This class holds a Get_PDU. The Get PDU is used for the HTTP/1.1 GET method.

Class Get_PDU

private Vector headers;
private HttpDecoder in;

GET_PDU(int tid, HttpDecoder in);
Reply_PDU getResponse();

7.5.1.1. Get_PDU:

This method is used to create a Get PDU header from the specified Tid and the HttpDecoder object. First of all there is URI length. Then comes the URI. Then there are headers. So in this way the Get PDU is obtained from the given HttpDecoder.

7.5.1.2. getResponse():

This method is there because the class extends the PDU abstract class which has this abstract method. This method sends the PDU to the remote Http-server and returns a response_PDU containing the response.

7.5.2. Class Post_PDU:

This class holds a Post_PDU. A Post PDU is also compliant to the HTTP/1.1 specifications. This PDU is used if the method used by the WAP client is Post type.

Class Post_PDU

private Vector headers;
private byte[] data;

Post_PDU(int tid, HttpDecoder in);
Reply_PDU getResponse();

7.5.2.1. Post_PDU(int, HttpDecoder):

The purpose of this method is exactly the same as that of Get_PDU() in the Get_PDU class. It creates a Post PDU. In a Post PDU first there is URI length, then comes the headers length, then comes the URI, then there is content type and at the end there are headers.

7.5.2.2. getResponse():

The reason for the existence of this method is due to the same reason as in the Get_PDU class. Here also it sends the PDU to the http-server and returns a response PDU that contains the response.

7.5.3. Class Reply_PDU:

This class is there to basically handle the response PDU which is to be sent to the WAP client. When a response comes from the http-server it cannot be sent directly to the WAP client. First it has to be encoded and then sent. So this class creates the reply PDU to be sent to client.

7.5.3.1. Instance Variables:

The instance variables defined in this class are all there in compliance with the HTTP/1.1 specifications. The response http packet contains all those fields.

7.5.3.2. answerError(int, String):

This method is used when the gateway discovers an error. This method constructs an answer with the status code as in the answer packet from the http-server.

```
Class Reply_PDU
-----------------------------------------
    private ByteVector status;
    private ByteVector hLen;
    private ByteVector contType;
    private ByteVector headers;
    private ByteVector data;
-----------------------------------------
private void answerError(int , String);
byte[] getByteArray();
Reply_PDU(int , HttpURLConnection);
Reply_PDU(int ,int, String);
Reply_PDU getResponse();
int size();
```

7.5.3.3. getByteArray():

This method is there because it converts all the fields of the response packet into the vector arrays. It makes use of the class ByteVector to convert the arrays into Vectors.

7.5.3.4. Reply_PDU(int , HttpURLConnection):

This method creates a Reply_PDU header from the specified TID and the HttpUrlConnection.

7.5.3.5. Reply_PDU(int, int ,String):

It creates a Reply_PDU with the specified startuscode and a textstring. The three parameters are as follows:

1. Transaction ID
2. HTTP status code
3. String to be displayed on the phone

7.5.3.6. Reply_PDU getResponse():

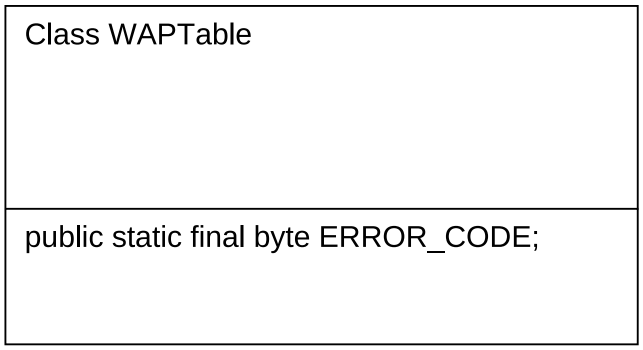
This method is there because of the reason that this class extends the abstract class PDU.

7.5.3.7. int size():

This method returns the size of the Reply_PDU in bytes.

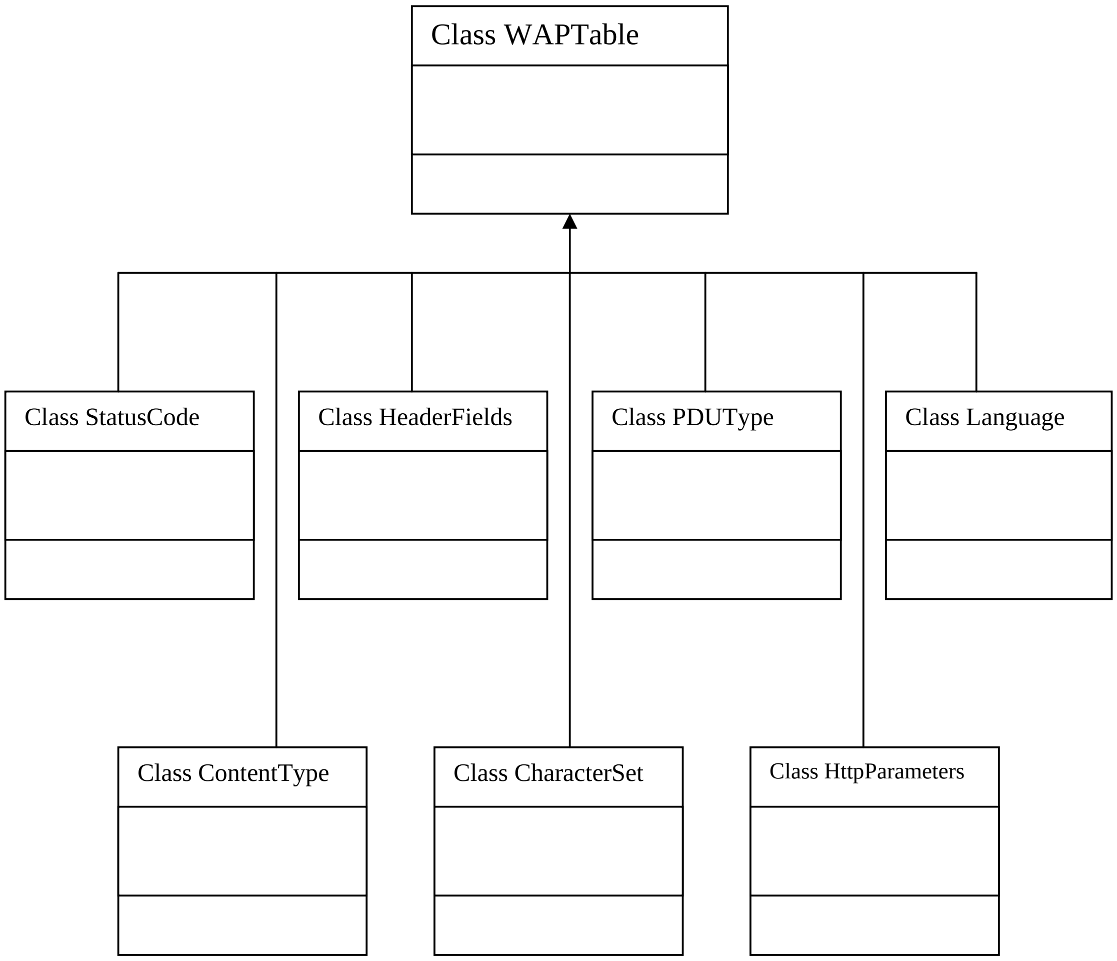
7.6 Class WAPTable:

This class is the super class of all the WAP Tables defined in the package. The class WAPTable defines the error code which is returned by it.



There are seven subclasses of this class. These classes are:

- CharacterSet
- ContentType
- HeaderFields
- Language
- PDUType
- HttpParameters
- StatusCode



7.6.1. Class StatusCode:

The status code class contains the information about all Http status codes. This information has been taken from the WAP WSP specifications and have been included as appendix at the end of this document.

Class StatusCode
private static final int[] HTTP_STATUS_CODE; private static final String[] STATUS_DESCRIPTION; private static final byte[] WAP_STATUS_CODE;

static int decode(byte);
static String describe(int status);
static String describe(int status);
static byte encode(int status);

7.6.2. Class HttpParameters:

This class contains the WSP encoding of the HTTP parameters. The information defined in this class has been taken from the WAP WSP specifications and has also been included at the end of this document.

Class HttpParameters
private static final byte[] PARAMETER_CODE; private static final String[] PARAMETER_NAME;

static byte getCode(String name);
static String getName(byte code);

7.6.3. Class CharacterSet:

This class contains the WSP encoding of the IANA MIBEnum values. The information defined in this class has been taken from the WAP WSP specifications and has also been included at the end of this document.

Class CharacterSet
private static final String[] CARACTER_SET; private static final int[] MIBENUM;
public static int getMIB(String); public static String getSet(int)

7.6.4. Class ContentType:

This class contains the WSP encoding of the HTTP content types. The information defined in this class has been taken from the WAP WSP specifications and has also been included at the end of this document.

Class ContentType
private static final String[] CONTENT_TYPE; private static final byte[] CONTENT_TYPE_CODE;

static byte getCode(String);
static String getName(byte);

7.6.5. Class Language:

This class contains the WSP encoding of the ISO 639 language assignment. The information defined in this class has been taken from the WAP WSP specifications and has also been included at the end of this document.

Class Language
private static final String[][] LANGUAGE; private static final byte[] LANGUAGE_CODE;

static byte getCode(String);
static String getFullName(byte);
static String getName(byte)

7.6.6. Class PDUType:

This class contains the WSP encoding of the Protocol Data Units. The information defined in this class has been taken from the WAP WSP specifications and has also been included at the end of this document.

Class PDUType
private static final String[] PDU_NAME; private static final byte[] PDU_NAME_CODE;
static byte getCode(String); static String getName(byte)

7.7 Class WMLCompiler:

This class is basically responsible for the parsing of WML document. It reads a WML document and compiles it into WML binary format.

Class WMLCompiler

private static final byte CHILD_BIT;
private static final byte ATTR_BIT;
private static ByteVector wmlcVector;
private static String wmlCharset;
private static String wbxmlCharset;

synchronized static public byte[] encode(InputStream , String, String);
private static ByteVector encodeAttrs(NamedNodeMap);
private static ByteVector encodeAttrValue(String);
private static void encodeTree(Node);
private static void init(String, String);
private static boolean isVarnameChar(char);
private static ByteVector makeText(String);
private static ByteVector makeVariable(String);

7.7.1. encode(InputStream , String, String):

This method encodes a wml document into its binary form. It throws NotWMLDocumentException if the wml document is not included inside the wml-tag. It throws WMLCParserException if the parser encounters an error. It returns an array of bytes containing the binary encoding.

7.7.2. encodeAttrs(NamedNodeMap):

This method encodes the attributes of an element used in the WML document.

7.7.3. encodeAttrValue(String):

This method encodes a ATTR_VALUE. Corresponding to each attribute there is an attribute value defined in the WSP specifications. So this method encodes the attribute value against the given attribute.

7.7.4. encodeTree(Node):

This method encodes a node in the DOM-tree and recursively encodes it's children.

Chapter 8
User Manual

In this chapter a detailed discussion regarding the use of the software is given. As we have implemented the WAP gateway using the Java language, which is platform independent so there is no constraint regarding the use of the software on any platform. The only constraint can be imposed by the WAP Browser or the Web Server. We have tested our applications on Windows 98, Windows 2000, Windows XP and WINNT. No problems have been encountered as far as the smooth running of the software is concerned.

8.1. REQUIREMENTS

Our software needs basically two more softwares to complete the WAP environment. As you already know that a WAP programming consists of three parts:

- WAP client.
- WAP gateway
- Web Server

In our application and demonstration we have used following softwares as WAP client and Web Server.

WAP Client	→	Ericsson WAP IDE 3.1
Web Server	→	LWS 2.2.1

8.2. Ericsson WAP IDE 3.1

WapIDE is a Software Development Kit (SDK) that enables operators, application developers, or any interested party to develop and test real WAP applications swiftly and easily. The main functions in WapIDE are:

- The *browser* simulates a WAP device and allows you to test WAP applications
  on different Ericsson phones.
- The *application designer* lets you create and test your own WAP applications.

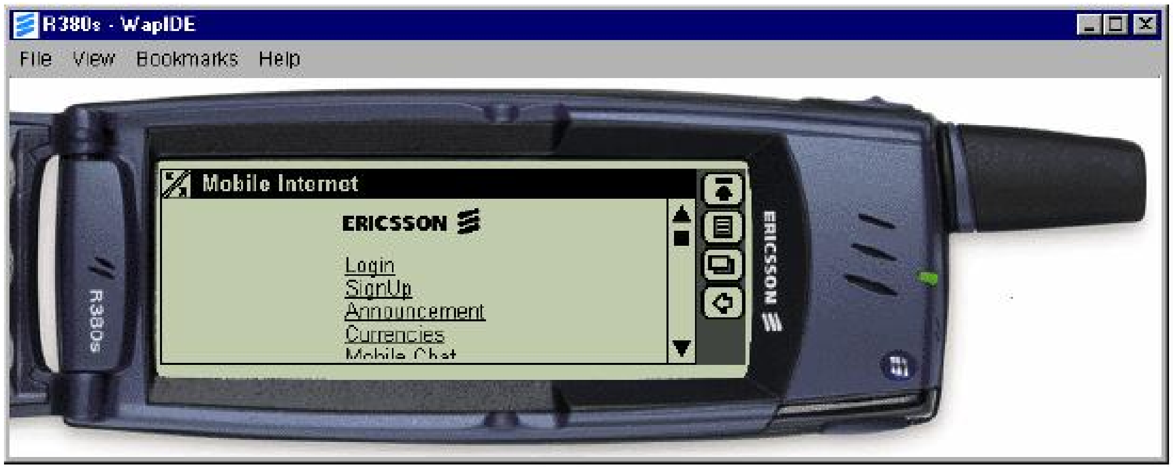
- The *push initiator* sends push messages to the WapIDE browser or a real terminal.

8.2.1. Browser

The WapIDE browser allows the user to access WML decks and cards using a Simulated WAP device. The following Ericsson devices are currently supported:

- R320s
- R380s
- R520m

The Chinese versions of these terminals are also indirectly supported since Chinese characters can be entered from the computer keyboard. The browser can access content from a web server via a *WAP gateway* or from a local disk.



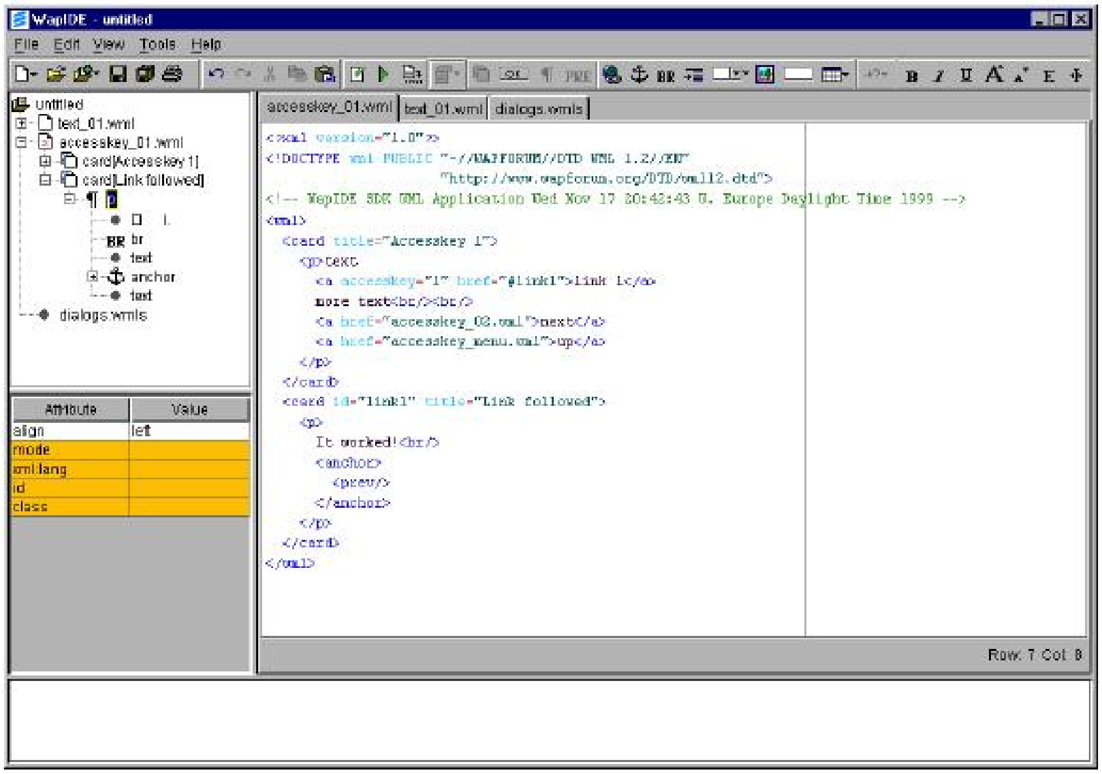
8.2.2. Application designer

The WapIDE application designer is a WML editor with which you can design and test WAP applications. There is also a WMLScript editor for writing and compiling

WMLScript code.

WML, WMLScript, and other files can be managed in projects.

The application designer is integrated with the WapIDE browser so you can easily test your applications on different devices.



Attribute	Value
align	left
mode	
xml:lang	
id	
class	

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
                    "http://www.wapforum.org/DTD/wml12.dtd">
<!-- WapIDE SDK WML Application Wed Nov 17 20:42:43 W. Europe Daylight Time 1999 -->
<wml>
  <card title="Accesskey 1">
    <p>text
      <a accesskey="1" href="#link1">link 1</a>
      more text<br/><br/>
      <a href="accesskey_02.wml">next</a>
      <a href="accesskey_menu.wml">up</a>
    </p>
  </card>
  <card id="link1" title="Link followed">
    <p>
      It worked!<br/>
      <anchor>
        <prev/>
      </anchor>
    </p>
  </card>
</wml>
```

8.2.3. Push initiator

The WapIDE push initiator is used to create and send push messages to the WapIDE
browser or a real terminal.

8.3. Using the WapIDE browser

The WapIDE browser is used to view WAP applications. It can be used instead of a WAP device to access WML decks developed by you or others. It also interprets WMLScript.

The browser can simulate different devices. It also supports applications in languages with different character sets, such as Chinese.

There are two ways to load content to the browser; from a:

- Local WML file
- Web server via a WAP gateway.

8.3.1. Starting the browser

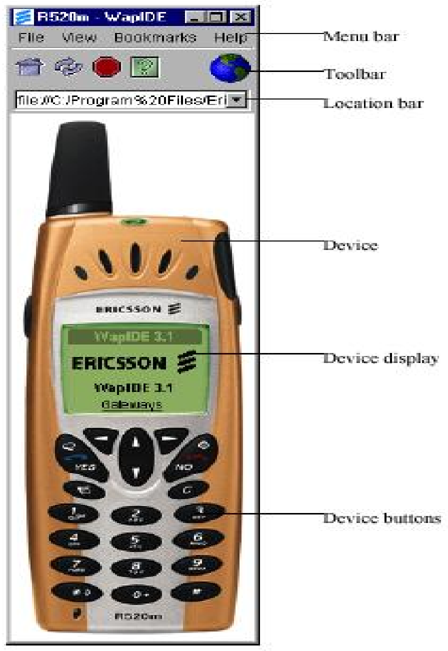
The browser can be started by selecting:

Programs → Ericsson WapIDE 3. (Browser

from the Windows Start menu.

The window shown on the next page is opened. Initially, the browser starts with the R520 device and the default home page (a local WML file).

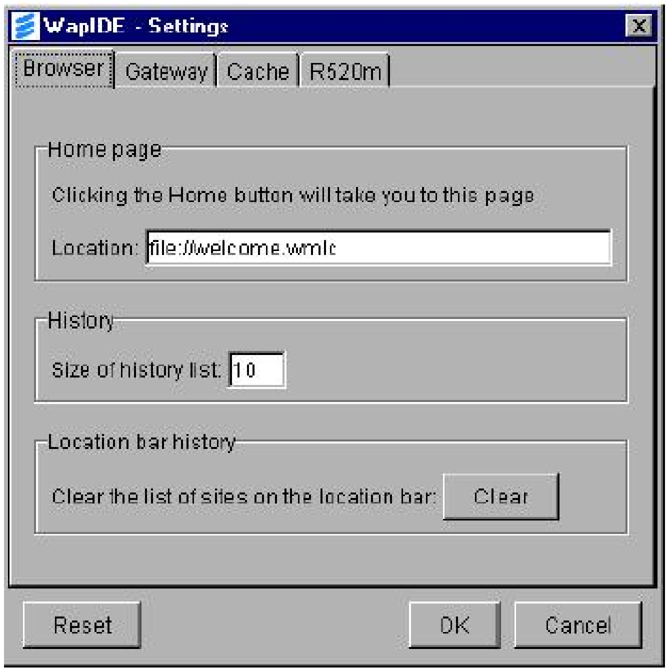
The default gateway is one that is available for external test use at the Ericsson Developers' Zone. To change the gateway, refer to Gateway settings.



8.3.2. Setting browser preferences

Selecting Settings from the View menu opens the settings window. There are four tabs in the window:

- Browser
- Gateway
- Cache
- Device-dependent



WapIDE - Settings

Browser | Gateway | Cache | R520m

Home page

Clicking the Home button will take you to this page

Location: file://welcome.wmlc

History

Size of history list: 10

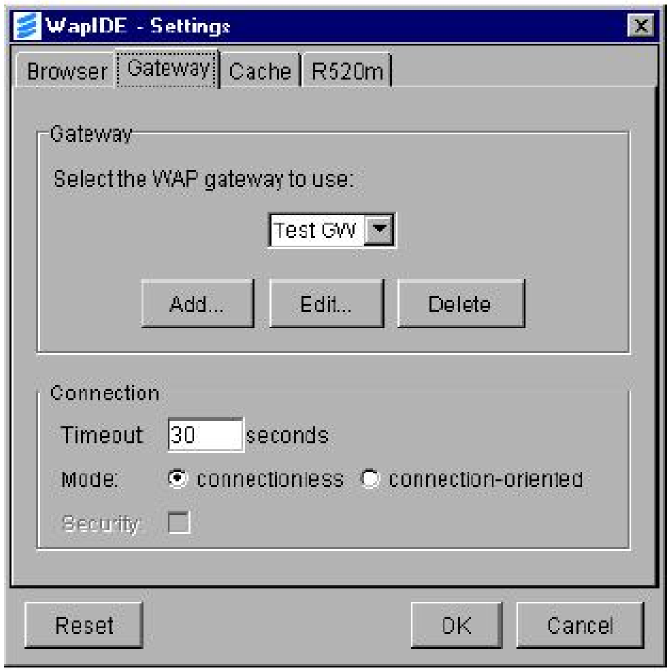
Location bar history

Clear the list of sites on the location bar: Clear

Reset | OK | Cancel

8.3.3. Gateway settings

Gateway Select the WAP gateway to use from the list. The default gateway is one that Ericsson provides for external test use (IP address 195.58.110.201). See the Ericsson Developers' Zone for more information.



Gateway

Select the WAP gateway to use: Test GW

Connection

Timeout 30 seconds

Mode: [✓ connectionless] connection-oriented

Security: ☐

Add...

When you add or edit a gateway, the following information can be entered:

Title - An optional description of the gateway. If not specified, the IP address is used.

IP address - The IP address of the gateway (required).

User ID/Password - Some gateways require a user ID and password.

Delete

Removes the selected gateway. Note that no confirmation window is shown.

Timeout: The number of seconds that WapIDE will wait for a reply from the gateway.

Mode: Use connectionless or connection-oriented sessions.

8.3.4. Accessing an application

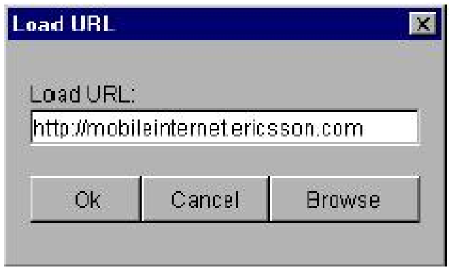
Applications are accessed by an URL. To load a URL, you can do one of the following:

- Type the address in the Location field as a normal Internet URL, e.g.:
  http://mobileinternet.ericsson.com
  You can also load a local WML-file by typing file://C:/path/file.wml

- Select an URL from the history list for the Location field.

- Select a bookmark.

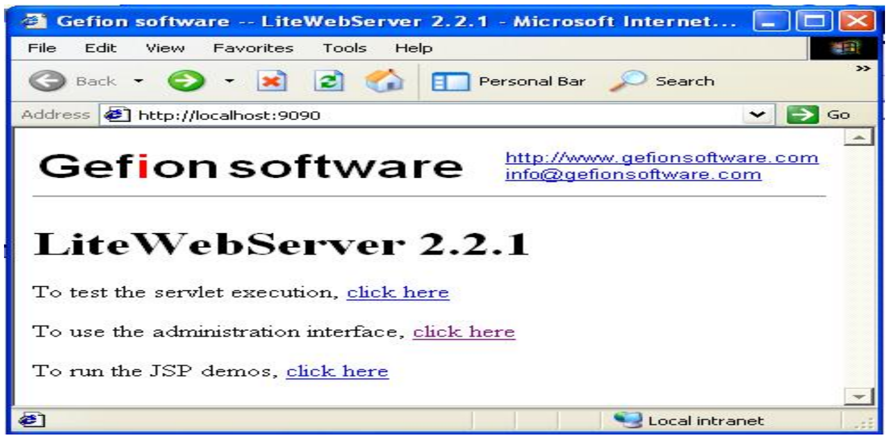
- Select **Load URL** from the **File** menu and type a URL in the same way as above.



Load URL: http://mobileinternet.ericsson.com

8.4. CONFIGURING THE WEB SERVER

After the settings for the gateway have been done, the next step is to configure the web server. For configuring the web server you have to add the MIME types required for the display of WML files. The MIME type for the WML file is text / vnd.wap.wml. This information has to be entered in the configuration file of the web server. Now the web server will be able to understand a WML page and display it.


Gefion software -- LiteWebServer 2.2.1 - Microsoft Internet...
File Edit View Favorites Tools Help
Back Personal Bar Search
Address http://localhost:9090 Go
Local intranet

Gefion software

http://www.gefionsoftware.com
info@gefionsoftware.com

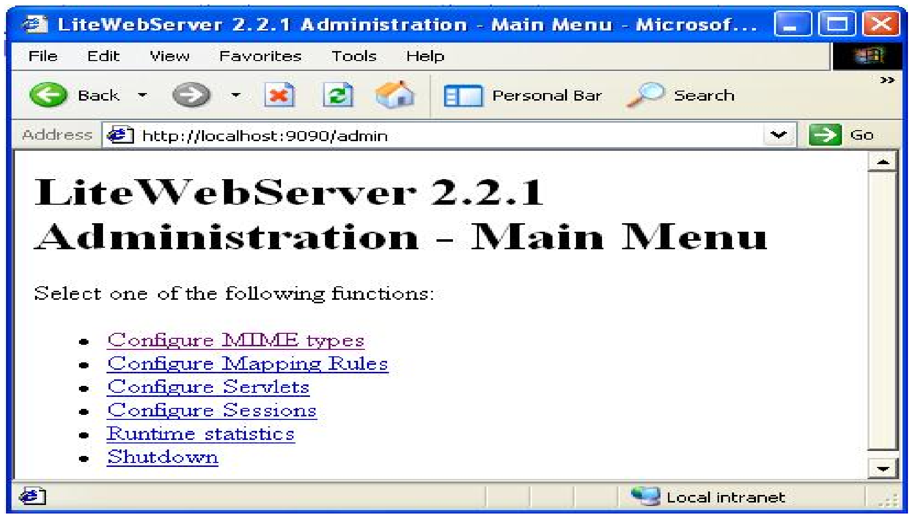
LiteWebServer 2.2.1

To test the servlet execution, click here

To use the administration interface, click here

To run the JSP demos, click here

Click on the administration interface to set the MIMIE types. It will open another link as follows:

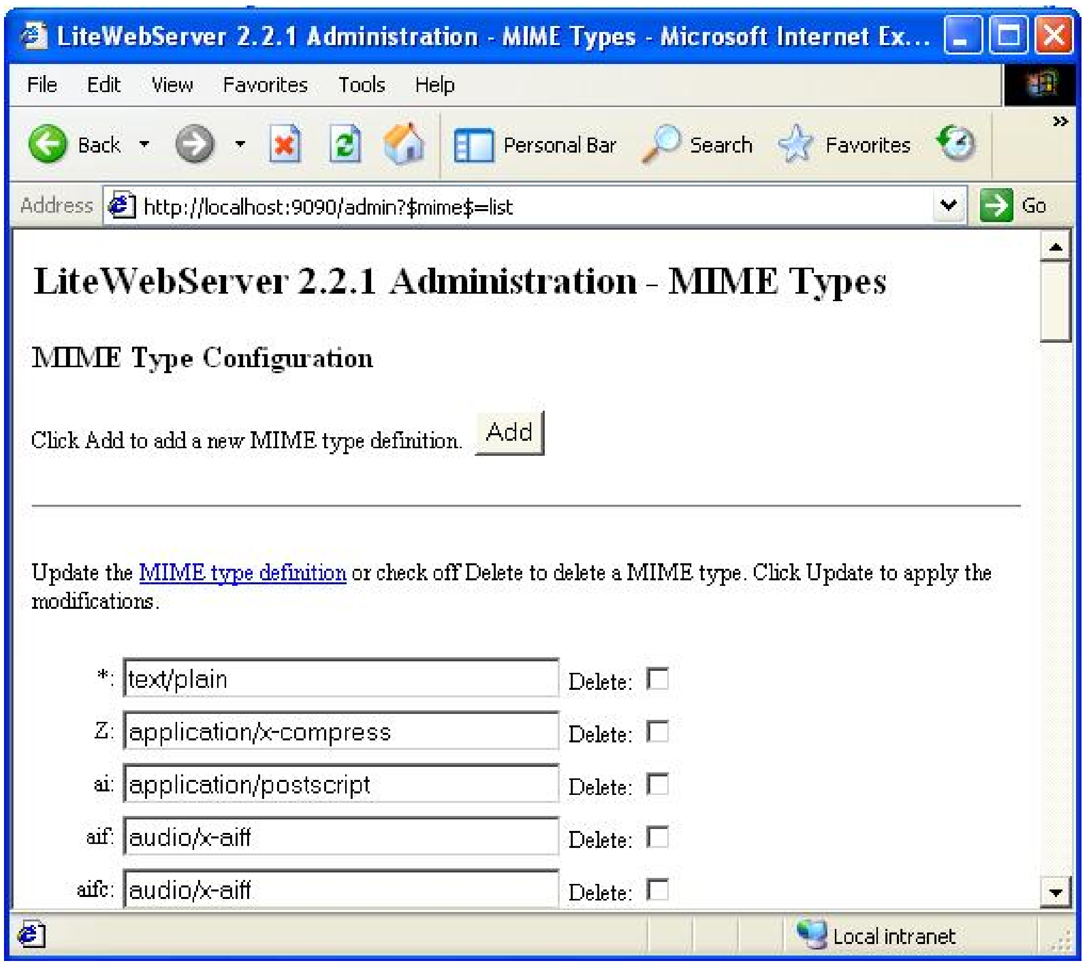

LiteWebServer 2.2.1 Administration - Main Menu - Microsof...
File Edit View Favorites Tools Help
Back Personal Bar Search
Address http://localhost:9090/admin Go
Local intranet

LiteWebServer 2.2.1
Administration - Main Menu

Select one of the following functions:

- Configure MIME types
- Configure Mapping Rules
- Configure Servlets
- Configure Sessions
- Runtime statistics
- Shutdown

Click on the option Configure MIME types and then add the particular MIME type into the web server settings.


LiteWebServer 2.2.1 Administration - MIME Types - Microsoft Internet Ex...
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost:9090/admin?$mime$=list
Go
Local intranet

LiteWebServer 2.2.1 Administration - MIME Types

MIME Type Configuration

Click Add to add a new MIME type definition. Add

Update the MIME type definition or check off Delete to delete a MIME type. Click Update to apply the modifications.

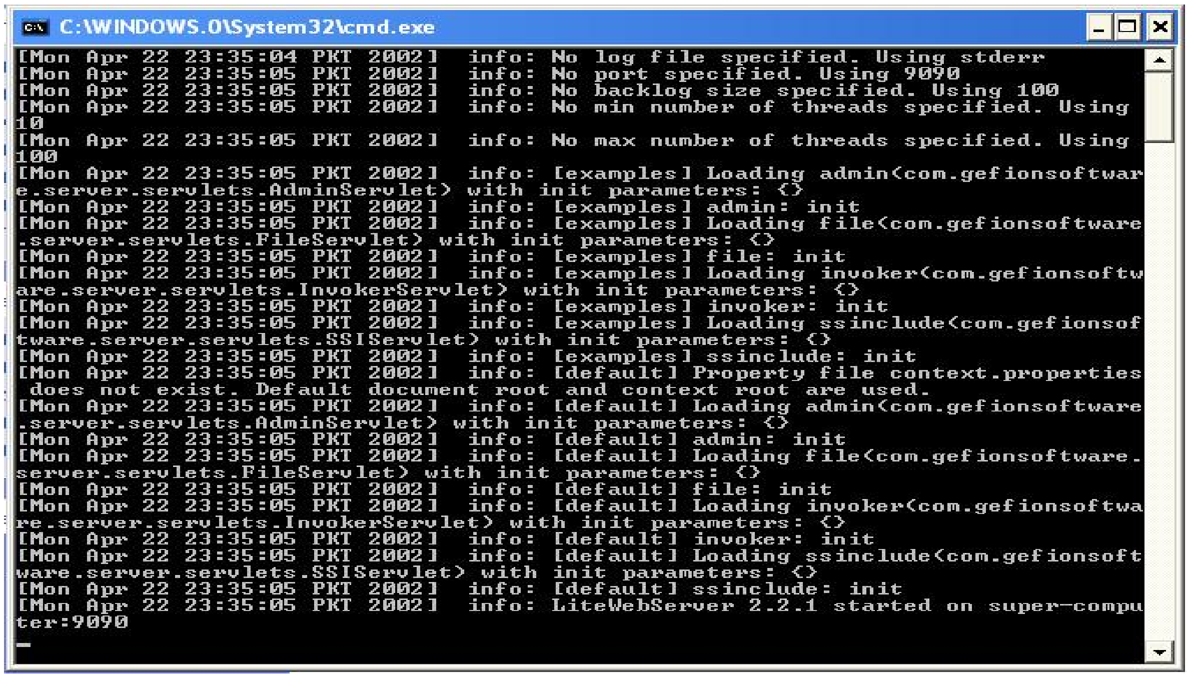
*:	text/plain	Delete: ☐
Z:	application/x-compress	Delete: ☐
ai:	application/postscript	Delete: ☐
aif:	audio/x-aiff	Delete: ☐
aifc:	audio/x-aiff	Delete: ☐

Now you have done the settings for both the WAP simulator and the Web Server and you are ready to test the Gateway.

8.5. WORKING OF THE GATEWAY

First of all start the web browser and also start the WAP simulator. Perform all the settings as described above because they are of most importance.

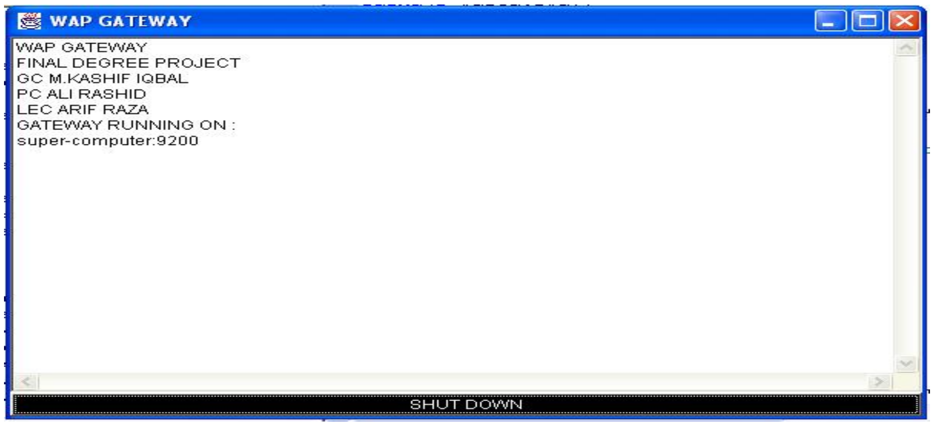
Once the web server is loaded the window appears as follows:



```
[Mon Apr 22 23:35:04 PKT 2002]  info: No log file specified. Using stderr
[Mon Apr 22 23:35:05 PKT 2002]  info: No port specified. Using 9090
[Mon Apr 22 23:35:05 PKT 2002]  info: No backlog size specified. Using 100
[Mon Apr 22 23:35:05 PKT 2002]  info: No min number of threads specified. Using
10
[Mon Apr 22 23:35:05 PKT 2002]  info: No max number of threads specified. Using
100
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] Loading admin(com.gefionsoftwar
e.server.servlets.AdminServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] admin: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] Loading file(com.gefionsoftware
.server.servlets.FileServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] file: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] Loading invoker(com.gefionsoftw
are.server.servlets.InvokerServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] invoker: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] Loading ssinclude(com.gefionsof
tware.server.servlets.SSIServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [examples] ssinclude: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] Property file context.properties
 does not exist. Default document root and context root are used.
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] Loading admin(com.gefionsoftware
.server.servlets.AdminServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] admin: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] Loading file(com.gefionsoftware.
server.servlets.FileServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] file: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] Loading invoker(com.gefionsoftwa
re.server.servlets.InvokerServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] invoker: init
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] Loading ssinclude(com.gefionsoft
ware.server.servlets.SSIServlet) with init parameters: {}
[Mon Apr 22 23:35:05 PKT 2002]  info: [default] ssinclude: init
[Mon Apr 22 23:35:05 PKT 2002]  info: LiteWebServer 2.2.1 started on super-compu
ter:9090
_
```

Now you start your WAP IDE so that you can view your applications on this simulator.

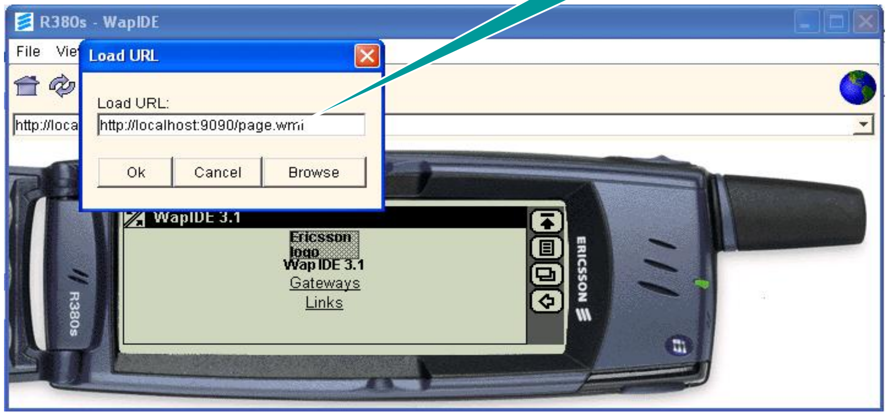
Once both have been started you can now start the WAP Gateway. A window will appear as follows:



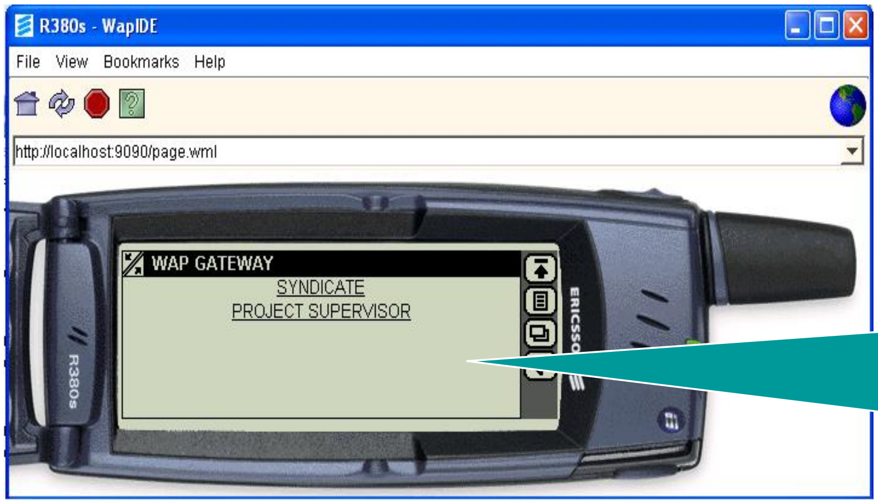
```
WAP GATEWAY
FINAL DEGREE PROJECT
GC M.KASHIF IQBAL
PC ALI RASHID
LEC ARIF RAZA
GATEWAY RUNNING ON :
super-computer:9200
```

Now load the URL from the WAP Simulator and a request will be made to the Gateway first and then to the web server. Consider the WAP Simulator accessing the following application.

URL TO THE DOCUMENT HAS BEEN

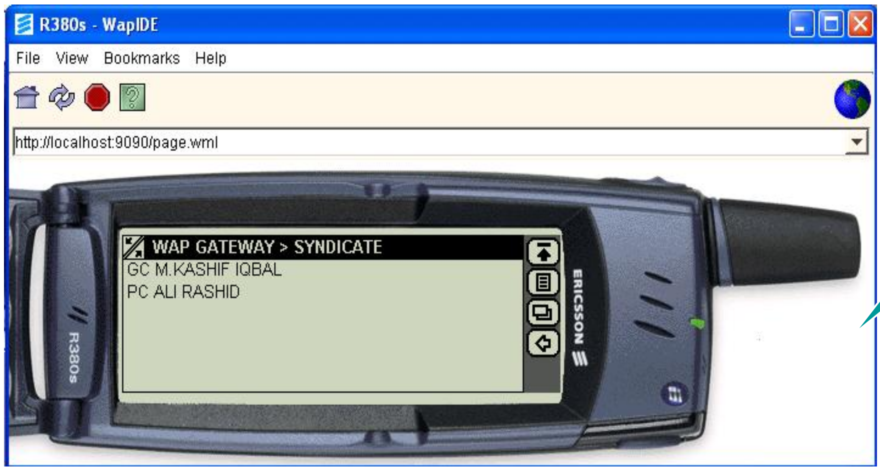


Now the request will go to the web server through the WAP Gateway and the WML page will be displayed on the simulator window.



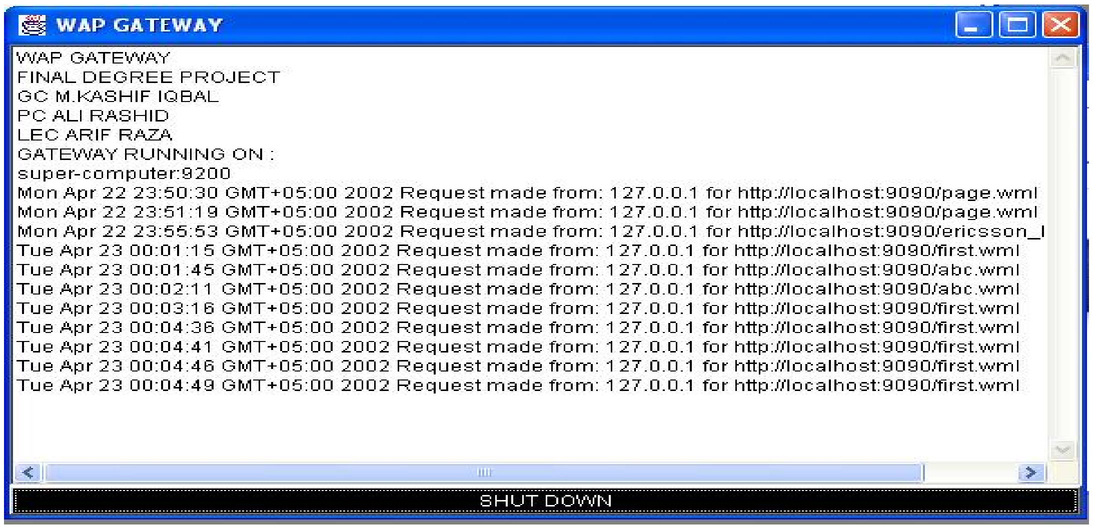
THE REQUESTED WML PAGE HAS BEEN ACCESSED BY THE CLIENT THROUGH THE GATEWAY

further links on the web page can be accessed by clicking on the links e.g.



ACCESSING FURTHER LINKS IN THE WML PAGE

The gateway will continue to display the messages for each request as follows:



```
WAP GATEWAY
FINAL DEGREE PROJECT
GC M.KASHIF IQBAL
PC ALI RASHID
LEC ARIF RAZA
GATEWAY RUNNING ON :
super-computer:9200
Mon Apr 22 23:50:30 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/page.wml
Mon Apr 22 23:51:19 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/page.wml
Mon Apr 22 23:55:53 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/ericsson_l
Tue Apr 23 00:01:15 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
Tue Apr 23 00:01:45 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/abc.wml
Tue Apr 23 00:02:11 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/abc.wml
Tue Apr 23 00:03:16 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
Tue Apr 23 00:04:36 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
Tue Apr 23 00:04:41 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
Tue Apr 23 00:04:46 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
Tue Apr 23 00:04:49 GMT+05:00 2002 Request made from: 127.0.0.1 for http://localhost:9090/first.wml
```

If you have to close the application you just click on the SHUT DOWN button and the Gateway will be closed.

Chapter 9
Summary and Future Recommendations

9.1. SUMMARY

The goal of our project was to build a WAP gateway in Java. The code is to be compliant with Java 1.1, because it is the most widespread Java platform of the available platforms.

The WAP protocol suite is available in several versions. The one implemented in mobile phones today is WAP 1.1 and to be able to communicate with the gateway from a real phone we had to use version 1.1 of WAP in our gateway.

We have implemented the connectionless part of the Wireless session protocol.

We have tested our application on Windows 98, Windows 2000, Windows XP and as such no problem has been encountered by us.

Our project was basically a simulation of the WAP Environment in which we have used the IP network as a bearer network. The actual environment of the WAP Gateway is using the GSM network as a bearer network and using a WAP enabled mobile phone. As at present no mobile operator is providing the facility of WAP so we were unable to test our application on the actual phone.

We have also created various applications for the demonstration of the gateway working. This includes various WML pages with several cards in it so that a clear picture of how WAP works is obtained.

Apart from that we have also built a servlet which when executes displays a list of songs and by selecting the song you can listen to mp3 songs.

9.2. RECOMMENDATIONS

On the basis of our study and project work we recommend the following additions, extensions and improvements to the existing WAP Gateway:

- Connection oriented part of WSP can be implemented.
- HTML to WML conversion can be added.
- Security features can be added to gateway.
- Domain name resolution (DNS) can be added.

9.3. CONCLUSION

Working on this project gave us a broad perspective about how to carry out a software project. Going through all stages of software development gave us a deep insight about the various developmental stages of a project. Moreover WAP was an entirely new field for us and a research in this field broaden our concepts regarding mobile communications, mobile devices, various constraints regarding the mobile communication, HTTP protocol and wireless markup language. It was really interesting to work min network programming. We have learned a lot after implementing the project and we can apply all this experience in our future practical life.

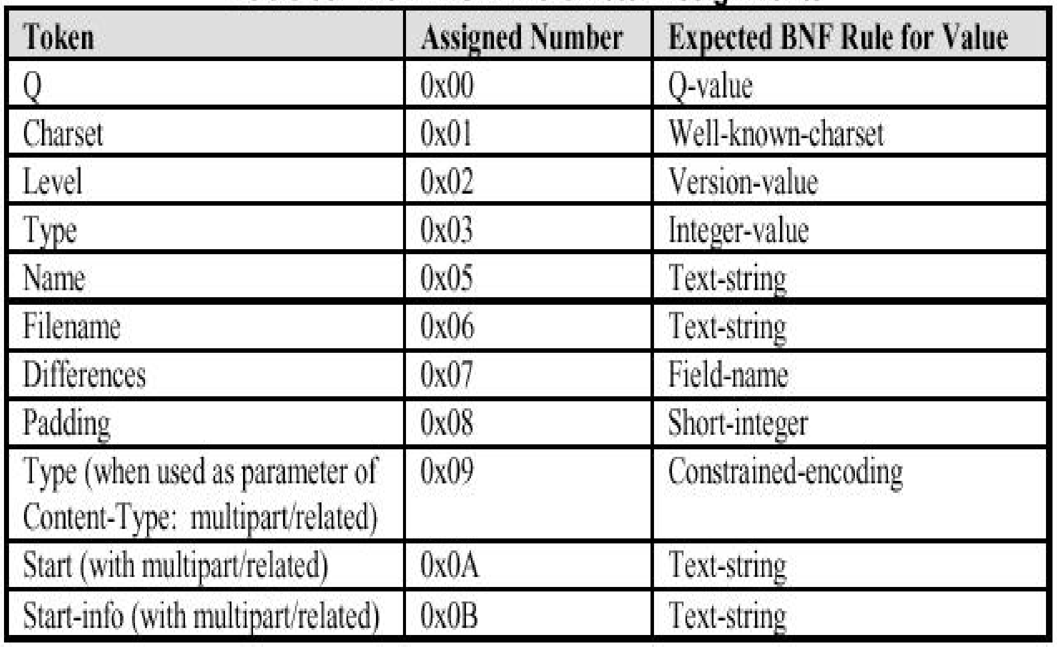
Chapter 10
Appendices

Appendix A
WSP Specifications

We have used the WAP WSP Specifications for the implementation of the WAP Gateway. Some of the Table that have been used by us include:

1. For the assignment of well known parameters of the http header following table from the WSP Specifications have been used.

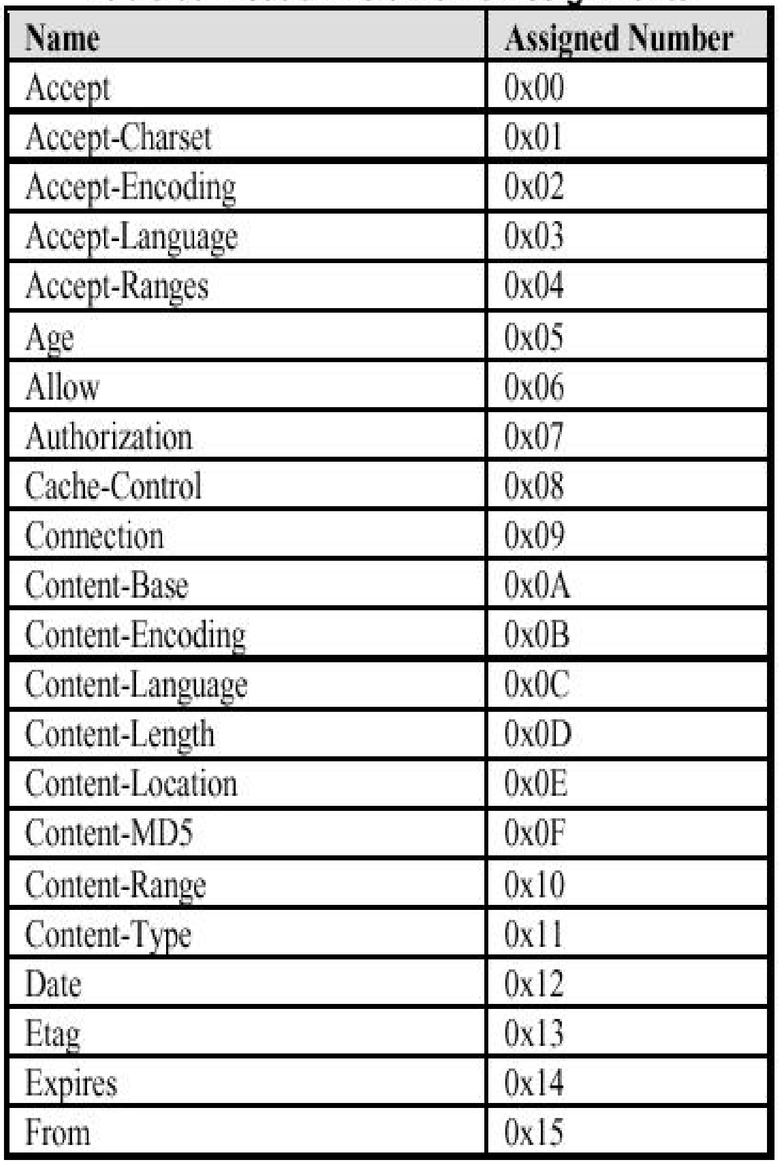
Table 38. Well-Known Parameter Assignments



Token	Assigned Number	Expected BNF Rule for Value
Q	0x00	Q-value
Charset	0x01	Well-known-charset
Level	0x02	Version-value
Type	0x03	Integer-value
Name	0x05	Text-string
Filename	0x06	Text-string
Differences	0x07	Field-name
Padding	0x08	Short-integer
Type (when used as parameter of Content-Type: multipart/related)	0x09	Constrained-encoding
Start (with multipart/related)	0x0A	Text-string
Start-info (with multipart/related)	0x0B	Text-string

2. For the header field name assignment following table have been used.

Table 39. Header Field Name Assignments



Name	Assigned Number
Accept	0x00
Accept-Charset	0x01
Accept-Encoding	0x02
Accept-Language	0x03
Accept-Ranges	0x04
Age	0x05
Allow	0x06
Authorization	0x07
Cache-Control	0x08
Connection	0x09
Content-Base	0x0A
Content-Encoding	0x0B
Content-Language	0x0C
Content-Length	0x0D
Content-Location	0x0E
Content-MD5	0x0F
Content-Range	0x10
Content-Type	0x11
Date	0x12
Etag	0x13
Expires	0x14
From	0x15

3. Following table have been used for the content type assignment of the WAP WSP values.

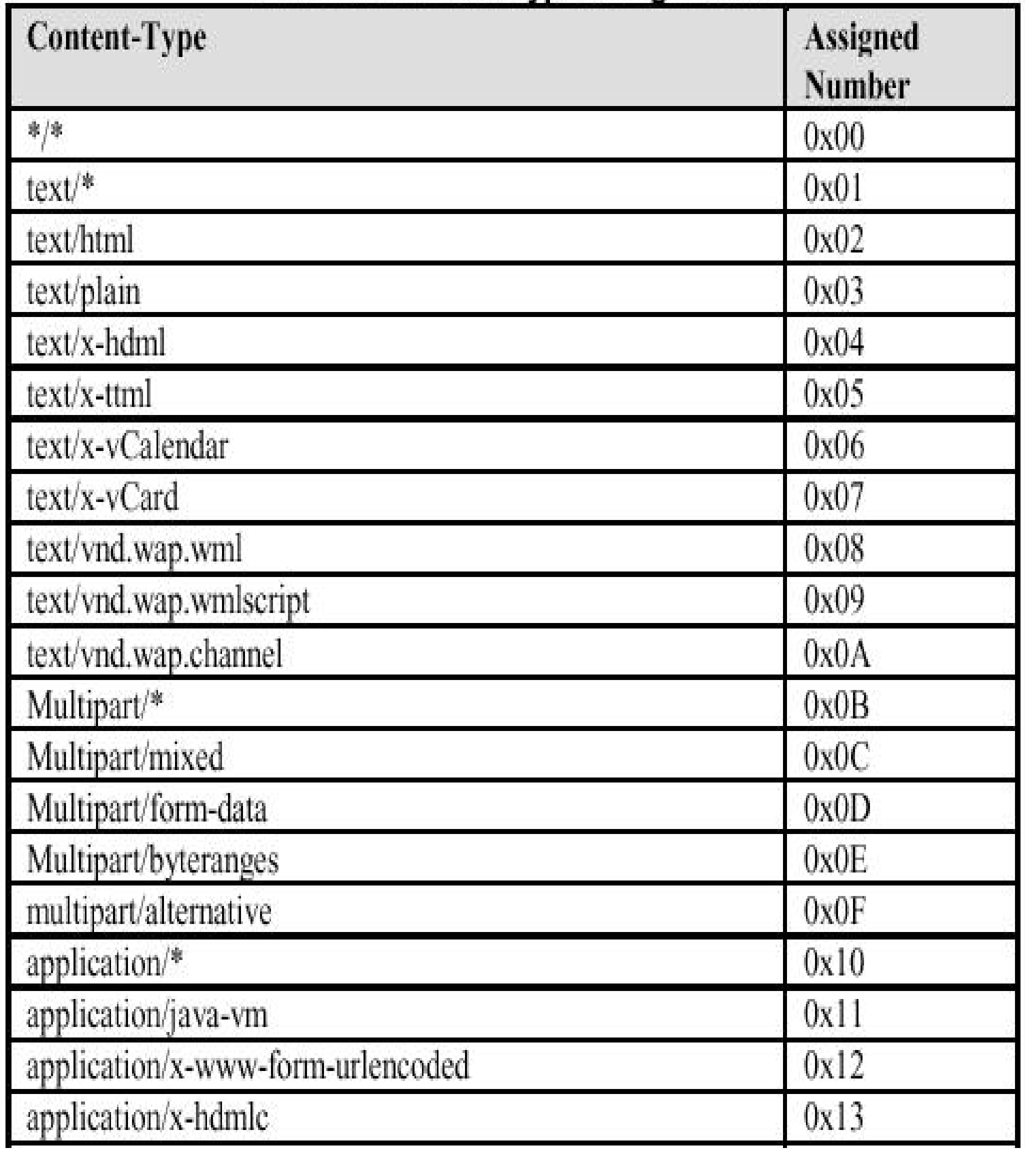
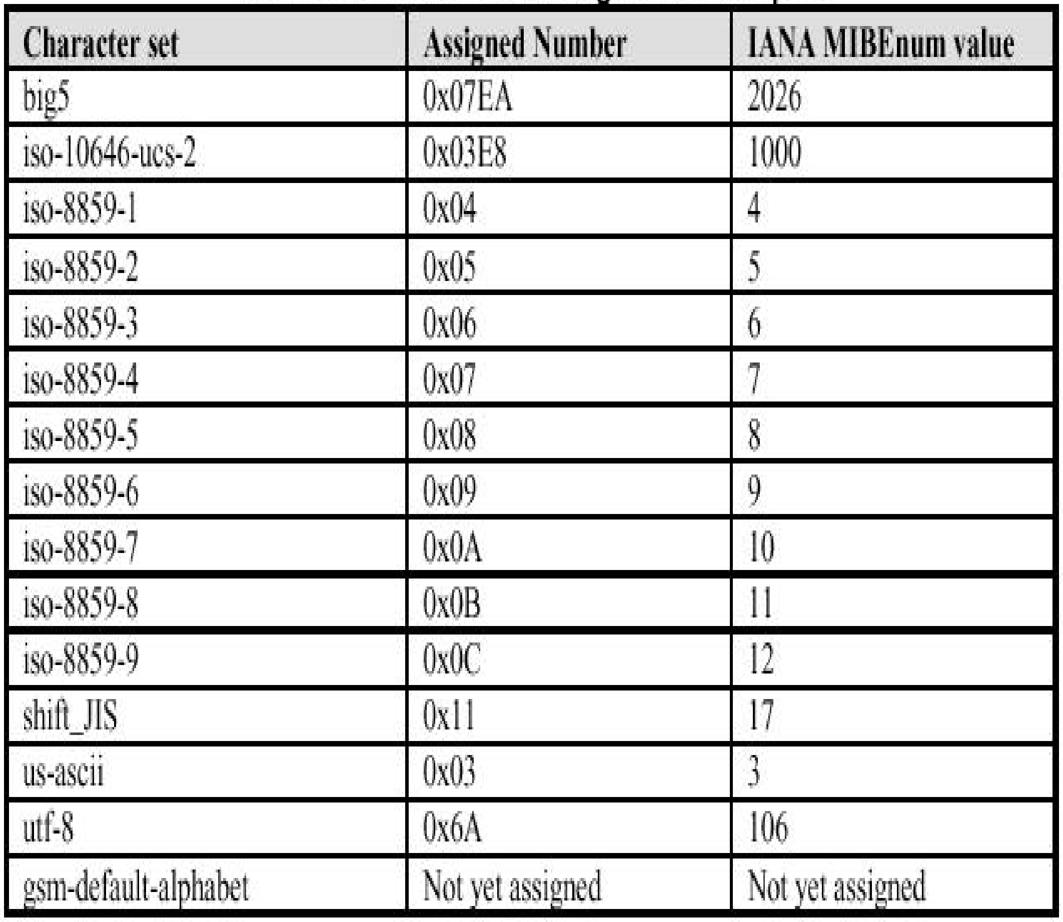


Table 40. Content Type Assignments

Content-Type	Assigned Number
/	0x00
text/*	0x01
text/html	0x02
text/plain	0x03
text/x-hdml	0x04
text/x-ttml	0x05
text/x-vCalendar	0x06
text/x-vCard	0x07
text/vnd.wap.wml	0x08
text/vnd.wap.wmlscript	0x09
text/vnd.wap.channel	0x0A
Multipart/*	0x0B
Multipart/mixed	0x0C
Multipart/form-data	0x0D
Multipart/byteranges	0x0E
multipart/alternative	0x0F
application/*	0x10
application/java-vm	0x11
application/x-www-form-urlencoded	0x12
application/x-hdmlc	0x13

4. Following table have been used for the character set assignment to the various character sets available.

Table 42. Character Set Assignment Examples



Character set	Assigned Number	IANA MIBEnum value
big5	0x07EA	2026
iso-10646-ucs-2	0x03E8	1000
iso-8859-1	0x04	4
iso-8859-2	0x05	5
iso-8859-3	0x06	6
iso-8859-4	0x07	7
iso-8859-5	0x08	8
iso-8859-6	0x09	9
iso-8859-7	0x0A	10
iso-8859-8	0x0B	11
iso-8859-9	0x0C	12
shift_JIS	0x11	17
us-ascii	0x03	3
utf-8	0x6A	106
gsm-default-alphabet	Not yet assigned	Not yet assigned

Appendix B Abbreviations

To make it easier for the reader to find the meaning of an abbreviation this list gathers all abbreviations used in this document.

- API – Application programming interface
- CGI – Common gateway interface
- HTML – Hyper text markup language
- HTTP – Hyper text transfer protocol
- IP – Internet protocol suite
- ISO OSI – ISO Open System Interconnection
- PDU – Protocol Datagram Unit
- PPP – point to point protocol
- SMS – short message service
- TCP – transmission control protocol
- TID – transaction identifier
- UDP – user datagram protocol
- URI – Uniform Resource Identifier
- WAE - Wireless Application Environment
- WAP - Wireless Application Protocol
- WML – wireless markup language
- WDP – wireless datagram protocol
- WBXML – wireless binary eXtensive Markup Language
- WTP – wireless transaction protocol
- WTLS – wireless transport layer security