**UNIT – I**

## CHAPTER 1: OVERVIEW OF WAP

- **WAP and the Wireless World**
- **WAP Application Architecture**
- **WAP Internal Structure**
- **WAP versus WEB**
- **WAP 1.2**

### OVERVIEW OF WAP

[WAP is] the de facto worldwide standard for providing Internet communications and advanced telephony services on digital mobile phones, pagers, personal digital assistants, and other wireless terminals - *WAP Forum.*

WAP stands for **W**ireless **A**pplication **P**rotocol. Per the dictionary definition for each of these words we have:

- **Wireless:** Lacking or not requiring a wire or wires pertaining to radio transmission.
- **Application:** A computer program or piece of computer software that is designed to do a specific task.
- **Protocol:** A set of technical rules about how information should be transmitted and received using computers.

WAP is the set of rules governing the transmission and reception of data by computer applications on or via wireless devices like mobile phones. WAP allows wireless devices to view specifically designed pages from the Internet using only plain text and very simple black-and-white pictures.

WAP is a standardized technology for cross-platform, distributed computing very similar to the Internet's combination of Hypertext Markup Language (HTML) and Hypertext Transfer Protocol (HTTP), except that it is optimized for:

- low-display capability
- low-memory
- low-bandwidth devices, such as personal digital assistants (PDAs), wireless phones, and pagers.

WAP is designed to scale across a broad range of wireless networks like GSM, IS-95, IS-136, and PDC.

**WAP and the Wireless World**

The WAP protocol is the leading standard for information services on wireless terminals like digital mobile phones.

The WAP standard is based on Internet standards (HTML, XML and TCP/IP). It consists of a WML language specification, a WMLScript specification, and a Wireless Telephony Application Interface (WTAI) specification.

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

**WAP Micro Browsers**

To fit into a small wireless terminal, WAP uses a Micro Browser.

A Micro Browser is a small piece of software that makes minimal demands on hardware, memory and CPU. It can display information written in a restricted mark-up language called WML.

The Micro Browser can also interpret a reduced version of JavaScript called WMLScript.

**What is WML?**

WML stands for **W**ireless **M**arkup **L**anguage. It is a mark-up language inherited from HTML, but WML is based on XML, so it is much stricter than HTML.

WML is used to create pages that can be displayed in a WAP browser. Pages in WML are called DECKS. Decks are constructed as a set of CARDS.

**What is WMLScript?**

WML uses WMLScript to run simple code on the client. WMLScript is a light JavaScript language. However, WML scripts are not embedded in the WML pages. WML pages only contains references to script URLs. WML scripts need to be compiled into byte code on a server before they can run in a WAP browser.

Visit our WMLScript tutorial to learn more about scripting in WML documents.

**Examples of WAP use**

- Checking train table information
- Ticket purchase
- Flight check in

- Viewing traffic information
- Checking weather conditions
- Looking up stock values
- Looking up phone numbers
- Looking up addresses
- Looking up sport results

## WAP – INTERNAL AND APPLICATION ARCHITECTURE

WAP is designed in a layered fashion, so that it can be extensible, flexible, and scalable. As a result, the WAP protocol stack is divided into five layers:

- **Application Layer**

  Wireless Application Environment (WAE). This layer is of most interest to content developers because it contains among other things, device specifications, and the content development programming languages, WML, and WMLScript.

- **Session Layer**

  Wireless Session Protocol (WSP). Unlike HTTP, WSP has been designed by the WAP Forum to provide fast connection suspension and reconnection.

- **Transaction Layer**

  Wireless Transaction Protocol (WTP). The WTP runs on top of a datagram service, such as User Datagram Protocol (UDP) and is part of the standard suite of TCP/IP protocols used to provide a simplified protocol suitable for low bandwidth wireless stations.

- **Security Layer**

  Wireless Transport Layer Security (WTLS). WTLS incorporates security features that are based upon the established Transport Layer Security (TLS) protocol standard. It includes data integrity checks, privacy, service denial, and authentication services.

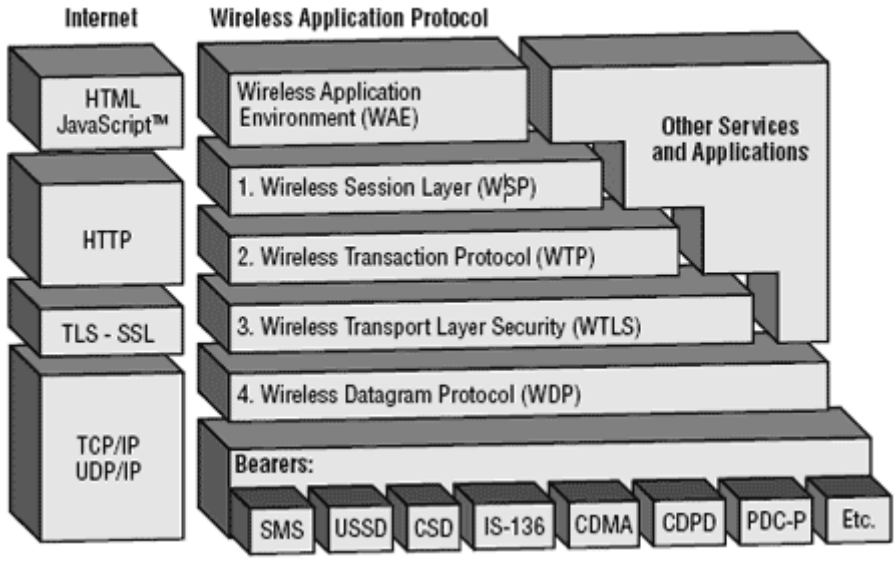
- **Transport Layer**

  Wireless Datagram Protocol (WDP). The WDP allows WAP to be bearer-independent by adapting the transport layer of the underlying bearer. The WDP presents a consistent data format to the higher layers of the WAP protocol stack, thereby offering the advantage of bearer independence to application developers.

Each of these layers provides a well-defined interface to the layer above it. This means that the internal workings of any layer are transparent or invisible to the layers above it. The layered architecture allows other applications and services to utilise the features provided

by the WAP-stack as well. This makes it possible to use the WAP-stack for services and applications that currently are not specified by WAP.

The WAP protocol architecture is shown below alongside a typical Internet Protocol stack.




Note that the mobile network bearers in the lower part of the figure above are not part of the WAP protocol stack.

**WAP 1.2**

In some circumstances, pushing content to a wireless device is preferable to letting the device's user pull content to them (for example, when an important e-mail arrives, or a goal is scored in a soccer game). Such push functionality is introduced in the WAP 1.2 specifications, and so, as yet, is not implemented in the devices currently available. However, in this section from Chapter 16 of Professional WAP (see Resources), we examine the framework behind push technologies and look at a simple example of a type of push mechanism using SMS.

**WAP VERSUS WEB**

**INFORMATION ON WEB**

The Internet model makes it possible for a client to reach services on a large number of origin servers, each addressed by a unique Uniform Resource Locator (URL).

The content stored on the servers is of various formats, but HTML is the predominant. HTML provides the content developer with a means to describe the appearance of a service in a flat document structure. If more advanced features like procedural logic are needed, then scripting languages such as JavaScript or VB Script may be utilized.

The figure below shows how a WWW client request a resource stored on a web server. On the Internet standard communication protocols, like HTTP and Transmission Control Protocol/Internet Protocol (TCP/IP) are used.

## INFORMATION ON WAP

WAP Gateway/Proxy is the entity that connects the wireless domain with the Internet. You should make a note that the request that is sent from the wireless client to the WAP Gateway/Proxy uses the Wireless Session Protocol (WSP). In its essence, WSP is a binary version of HTTP.

A markup language - the Wireless Markup Language (WML) has been adapted to develop optimized WAP applications. In order to save valuable bandwidth in the wireless network, WML can be encoded into a compact binary format. Encoding WML is one of the tasks performed by the WAP Gateway/Proxy.

## WTA and PUSH Features

Assigning style rules to the root element

You do not have to assign a style rule to each element in the list. Many elements can rely on the styles of their parents cascading down. The most important style, therefore, is the one for the root element — SEASON in this example. This defines the default for all the other elements on the page. Computer monitors at roughly 72 dots per inch (dpi) don't have as high a resolution as paper at 300 or more dpi.

Therefore, Web pages should generally use a larger point size than is customary. Assigning style rules to the root element

You do not have to assign a style rule to each element in the list. Many elements can rely on the styles of their parents cascading down. The most important style, therefore, is the one for the root element — SEASON in this example. This defines the default for all the other elements on the page. Computer monitors at roughly 72 dots per inch (dpi) don't have as high a resolution as paper at 300 or more dpi.

Therefore, Web pages should generally use a larger point size than is customary.

## Wireless Telephony Application (WTA)

o Collection of telephony specific extensions

o Extension of basic WAE application model ν content push

o server can push content to the client

o client may now be able to handle unknown events ν handling of network events o table indicating how to react on certain events from the network ν access to telephony functions (WTAI)

o any application on the client may access telephony functions o Example ν calling a number (WML) wtai://wp/mc;07216086415 ν calling a number (WMLScript) WTAPublic.makeCall("07216086415");

**PUSH FEATURES**

**WAP push architecture with proxy gateway**

o Push Access Protocol ν Content transmission between server and PPG ν First version uses HTTP o Push OTA (Over The Air) Protocol ν Simple, optimized ν Mapped onto WSP Client User Agents Push Proxy Gateway Coding, checking Push OTA Protocol Push Initiator Push Access Protocol Server application

**CHAPTER 2: SETTING UP WAP**

- **Available Software Products**
- **WAP Resources**
- **The Development Toolkits**

**Available Software Products and WAP resources**

- WAP Forum - Official website of WAP Forum. Find all the latest news, inventions and standards available at this site.
- WAP technical specifications – Latest technical specification of WAP technology.
- WinWAP simulator – Download WinWAP browser from their official website.
- WAP Gateway – Download WAP gateway from this site.
- WAP phone vendors - Nokia – One of the best WAP Phone venodrs
- WAP phone vendors - Ericsson – One of the best WAP Phone venodrs
- GPRS @ Wikipedia – A very useful article on GPRS technology and further more links to useful sites.
- GSM @ Wikipedia – A very useful article on GSM techinology and further more links to useful sites.
- GSM Arena – A site providing comprehensive reviews, feedback, and prices of all the available GSM Mobile Models in the market.
- **WAP Forum** - Very good site for all wap resources, nice forum for related technologies

- **Wap.com** - One place for wireless internet, wap phones and all the related technologies.
- **Wireless Developer Network** - Excellent site for wireless developers. Articles, new, tools and much more. Also contains information about related technologies such as Blue tooth, GPRS, Jini and so on.
- **Wireless in a Nutshell** - Another good source of information and tools.
- **Nokia** - A good site for developers looking for tools, wml documentation, device specific information, wap enabled phones, tags they support, Nokia WAP SDK and much more. Another important link on this site is the Nokia forum.
- **Ericsson** - Another good site, similar to Nokia, but with Ericsson specific information and tools. Another important links on Ericsson site are developer's zone and wap section.
- **Yospace** - Mainly emulators, I believe they have made the best emulator for Nokia 7110 WAP phone. Excellent emulators for many other WAP enabled devices such as Nokia(7110, 6210), Ericsson(R320s, R380) and others through their Smart Phone emulator. Also good section on wireless games.
- **Wap Resources** - A very good source of information on wml, wmlscript, wbmp, browsers, gateways, source code samples, SMS, iMode, 3G, Bluetooth and so on.
- **Wap Drive** - They host WAP sites for *free*. Have already hosted over 100,000 wap sites. Interested???
- **Wapsilon** - They proved nice online emulator for wap sites, so you can let your visitors have a look at your wap site without a wap enabled device, using a normal web browser.
- **WBMP Gallery** - This is the largest *free* wbmp library on the web. Amazing, how they made so many wonderful images in wbmp format, which uses only 2 colors. You can use all these wbmp for your wap pages for free. Classified nicely into various categories.

## THE DEVELOPMENT TOOLKITS

### A First Look At WAP Toolkits From Nokia, Ericsson, and Phone.com

While still in its infancy, the Wireless Application Protocol has certainly generated a tremendous amount of interest. As WAP services are already beginning to appear in Europe and many more are on their way in North America and Asia, a variety of WAP vendors have also made available development tools that enable WAP application development and deployment. Please note that this is not a complete WAP tutorial. You can visit our WAP Training section for that information. In this review, we will examine three leading WAP development toolkits:

- Ericsson WapIDE 2.0
- Nokia WAP Toolkit 1.2
- Phone.com UP.SDK 4.0

While all three companies were instrumental in the formation of the WAP Forum, it is interesting to note that their development tools differ in many ways. It is also important to note that each of these companies sells commercial WAP servers which would probably be used in conjunction with their respective toolkit. We will delay a review of commercial WAP servers for a later issue.

***Ericsson WapIDE 4.0***

The Ericsson WapIDE product consists of a suite of tools that support the design and testing of WAP applications as well as the ability to design a new WAP device in order to test the device for look-and-feel issues. The WapIDE Software Development Kit is currently only available for Windows NT 4.0 and Windows 95/98. Included with the IDE are products for testing server applications. These products include Perl 5.0, Tcl/Tk, and the Xitami Web Server. Installing the WapIDE first requires the installation of the IDE followed by the installation of the SDK. I attempted to opt out of the Xitami Web Server installation (since I'm already running Microsoft IIS and Apache on my machine!) but the installation apparently failed at that point. I suppose one never knows when they might need three HTTP servers on one laptop! Figure 1 shows the WapIDE interface to the tools including the Browser (used to test applications), the App Designer (used to construct applications), and the Server Toolset (a set of tools including a WML/WMLScript compiler and a Syntax Analyzer).

Figure 1 - The WapIDE Interface

It's not clear to me why the Server tools weren't simply included in the App Designer. It seems a bit disjointed to jump back and forth between applications in order to build code and then compile the code. The Browser (see Figure 2) supports the use of different devices (the R320s is the default) and allows the user to test the sample WAP URLs included with the toolkit or to test their own creations. A variety of sample WAP applications are included with the IDE for banking, stock quotes, and schedules.

Figure 2 - The WapIDE Browser

The bulk of your time will be spent in the AppDesigner application (Figure 3). This tool integrates a WML editor with the WapIDE browser that coding and testing can be done within one application. It's a bare-bones development environment with no frills. The documentation is also a bit sparse although quite a few Adobe Acrobat documents are available for download from the Ericsson WAP Developer site.

Figure 3 - The WapIDE AppDesigner

Other WAP products from Ericsson include the WAP Application Server (a Solaris-based Java application server designed to scale to 50,000 users that allows developers to build device-independent applications) and the WAP Gateway (an Intel platform-based product that acts as a server to a GSM network and bearers such as SMS and USSD).

***Nokia WAP Toolkit 1.2***

The WAP Toolkit 1.2 product from Nokia is similar in some respects to Ericsson's WapIDE. Both products contain graphical development environments (though neither one supports any type of of drag-and-drop UI creation), browsers, and WML/WMLScript compilers. The Nokia toolkit currently runs only on Windows NT 4.0 but note that the Nokia WAP Toolkit also requires a Java 2 runtime. You will want to make a visit to Sun's Java Web site to download either the Java 2 SDK or the Java 2 Runtime Environment (JRE) before evaluating the WAP Toolkit product.

After installation, the WAP Toolkit Program Group (under Windows) will contain shortcuts to the toolkit Integrated Development Environment (IDE) as well as excellent documentation on WAP, WML, WMLScript, and the toolkit itself. The toolkit application itself (see Figure 4) supports the creation, modification, and testing of WML/WMLScript code within one application.

Figure 4 - The Nokia WAP Toolkit

The user can toggle between loading the WAP applications through HTTP or via a WAP gateway. Nokia also sells a separate Java Servlet-based WAP Server product. This Server incorporates the application server and WAP gateway functionality into one product. In all, the Nokia WAP products appear to be well-thought out and functional and is superior, from a user interface standpoint, to the Ericsson WapIDE product.

***Phone.com UP.SDK 4.0***

The Phone.com UP.SDK product (available for Windows 95/98/NT and Solaris) differs a bit from the Nokia and Ericsson product in that no graphical IDE is provided with the product. Instead of focusing on providing an integrated environment for editing and testing WML/WMLScript code, UP.SDK focuses much more heavily on providing a set of reusable code libraries for use with languages such as WML, Perl, C, C++, and Visual Basic. As Phone.com is the manufacturer of the leading WAP microbrowser, naturally the UP.SDK comes with a WAP browser known as the UP.Simulator (see Figure 5). Note that the Simulator is currently only available for the Windows platform.

Figure 5 - The Phone.com Up.Simulator

You will need a live Internet connection because the simulator actually dynamically connects to the Phone.com developer Web site. It does this in order to download samples and access live WAP applications on the Web. I definitely recommend checking this product out because it will give you a good feel for how WAP can be used (I was able to check my local weather and favorite stock quotes in a manner of seconds using the Up.Simulator). The UP.SDK also includes Perl and C library functions for generating WML and handling HTTP requests as well as C++ (Solaris) and COM (Windows) objects for notification, digest, and fax handling. The

UP.SDK also includes tools for requesting and installing SSL certificates for security purposes. Besides providing the standard WAP functionality, Phone.com extends WAP's capabilities through fax and notification support. The UP.LINK server includes a Fax Manager product which allows handheld users to fax information directly from their WAP browser! Postscript, ASCII text, Microsoft Word, RTF, and Adobe Acrobat document types are accepted as fax or fax response formats. Asynchronous notifications can also be *pushed* to handheld clients via the Phone.com Notification API. This API allows the control of alerts, document cache, and decks on the client.

***Conclusion***

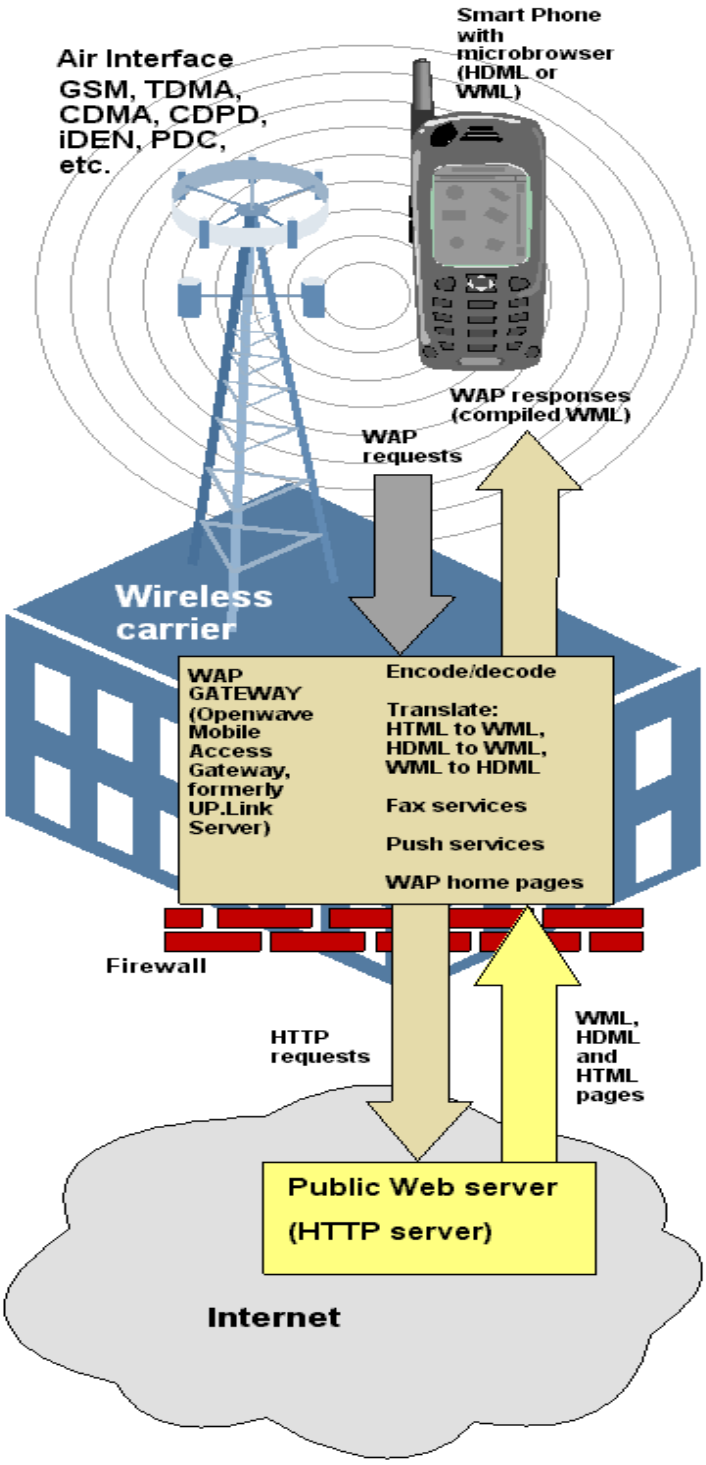
If you are setting out to develop your first WAP application, the Nokia WAP Toolkit product supplies excellent documentation as well as an easy-to-use graphical development environment. Meanwhile, the Phone.com product offers a great deal more capabilities but no user-friendly development tools. Of the three products, I would say that the Ericsson WapIDE tool is still a bit rough around the edges and could use some polishing. If you're interested in WAP development, be sure to download all three of the products from the links below. Exploring the tools and building examples with them is a great way to get started learning WAP!

# UNIT – II

## WAP gateway

(**W**ireless **A**pplication **P**rotocol gateway) Software that decodes and encodes requests and respons es betweenthe smartphone microbrowsers and the Internet. It decodes the encoded WAP requests f rom the microbrowser and sends the HTTP requeststo the Internet or to a local application server. I t encodes WML and HDML data returning from the Web for transmission to the microbrowserin the handset. See **WAP**.

| **A WAP Gateway** |
| --- |
| This diagram depicts Openwave's Mobile Access Gatewaywhich combines the WAP gateway with an applicationserver that hosts WAP pages and provides numerousservices (push, fax, et c.) along with encoding anddecoding. |


Smart Phone with microbrowser (HDML or WML)
Air Interface
GSM, TDMA, CDMA, CDPD, iDEN, PDC, etc.
WAP responses (compiled WML)
WAP requests
Wireless carrier
WAP GATEWAY (Openwave Mobile Access Gateway, formerly UP.Link Server)
Encode/decode
Translate: HTML to WML, HDML to WML, WML to HDML
Fax services
Push services
WAP home pages
Firewall
HTTP requests
WML, HDML and HTML pages
Public Web server
(HTTP server)
Internet

**FUNCTIONALITY OF A WAP GATEWAY**

**WAP -> Gateways**

The simplest way to think about a WAP gateway is that it is the system that WAP devices communicate directly with, and any content delivered to WAP devices via IP networks (like the Internet) must travel through these gateways on their way to those little, tiny screens.

The WAP gateway performs a host of duties, which we'll cover here in detail, but the most important is the protocol translations that are necessary to connect the TCP/IP-based world of the wired Internet with the WAP-based world of wireless mobile devices. Without something to perform this essential task, WAP simply wouldn't work.

A WAP gateway is a piece of software that usually resides on a dedicated Unix-type of server. Often the box itself will also be referred to as the gateway, but for the purposes of this article we will only be considering the software functionality of WAP gateways. There are also WAP gateways that can run on Windows-based systems, as well as combination content-server/gateway products, but the norm is to have a stand-alone Unix server running the gateway software

Functionality of a WAP Gateway

1.Implementation of "WAP protocol stack" layers.

2.Protocol conversion: WSP HTTP.

3.Domain Name resolution.

4.HTML to WML conversion.

5.Encoding of WML Content.

6.WMLScript compilation.

7.Security.

8.Caching for frequently accessed content.

1.Implementation of WAP protocol stack layers :-

Depending on whether the type of service is connection –oriented or connectionless, secure or

not secure, the following layers need to be implemented:

a) Secure connection –oriented : WAE–WSP-WTP-WTLS-WDP

b) Secure connectionless : WAE-WSP-WTLS-WDP

c) Non secure connection –oriented : WAE–WSP-WTP-WDP

d) Non Secure connectionless : WAE–WSP-WDP

2.Protocol conversion :

WSP HTTP : WSP supports complete HTTP 1.1, this include

a) request-reply methods : Get, post,… etc

b) request

c) response

d) Entity headers like "Accept: application/vnd.wap.wmlc"- request header that specifies the MIME types that a client can handle.

e) Content negotiation –is the process of selecting the best representation for a client for a given response when there are multiple representations for the same content available from the server.

3.Domain name resolution.

• Resolution of domain names, used in URLs, to IP address is done by domain name server (DNS) this is optional if the Gateway uses an HTTP proxy to retrieve the content, in the case of public internet HTTP proxy has the responsibility of resolving internet domain names instead of WAP Gateway in this case.

4.HTML to WML conversion

a) It is optional feature, this conversion can never be perfect, and it will not be rendered probably on a wireless device.

b) Content providers actually provide WML content separately.

c) Content providers provide WML & HTML contents in the same server, and the server looks at the"user-Agent" and /or "Accept:" HTTP request header to decide which one to send.

d) Provide content in XML and convert this to HTML or WML using XSLT for the transformation.

5.Encoding of WML content :-

• This process is known as tokenization, during this process, the Gateway checks to verify that the WML content has no errors and is well formatted (because WML is XML language )

**What does a Gateway Do?**

A WAP Gateway plays many roles in the scheme of turning the WAP model into working services. A list of just some of the functions of a WAP gateway include the following:

- implementation of the WAP stack
- converting protocols
- converting markup languages
- compiling WMLScript programs
- encoding WML into a binary bitstream
- providing access control
- caching
- domain name resolution services (DNS)
- security features

Clearly, these gateways can pack a lot of punch. There is a wide range of products available, with differing feature sets and price ranges (from free to big bucks) so we'll deal with the most commonly implemented and important features here.

Providing the software to maintain the WAP stack layers is a critical function of any WAP gateway, though they will not all implement the same layers. Gateways that support connectionless services will not need to implement the WTP layer, just as products that don't support secure services won't need to include the WTLS layer. A gateway that will be used for secure, connection-oriented services will need to implement WSP, WTP, WTLS, and WDP.

Converting between WAP's WSP and the Internet's HTTP is another central function of any WAP gateway. One of the main differences between WSP and HTTP is that WSP's headers are encoded in binary form, or "tokenized." For example, the token "Accept" from the HTTP/1.1 header becomes "0x80". This is a way for WAP to save on bandwidth by transmitting the bare minimum bits possible to represent a specific group of characters.

Some gateways also offer HTML to WML conversion, though this feature rarely results in functional WAP services. It is generally agreed that mobile applications should serve WML directly from the origin server, and that the code needs to be designed and tested separately from

HTML implementations.

Another important function of the WAP gateway is the compilation of WMLScript programs. WMLScript is the only scripting language that works with WAP devices, and although there are many similarities to Javascript, there are some key differences in how it is called and compiled. Where Javascript code is enclosed inside of HTML files and interpreted by the client browser, WMLScript code is put in a separate file, and compiled at the gateway, in the constant quest to reduce processing requirements at the device end. The compiled script is then sent to the device as a binary encoded stream, also reducing bandwidth.

Encoding the WML content is another mandatory function of a gateway, and again it is done to save on the precious bandwidth of the wireless networks. By converting the WML to a binary, tokenized format, the amount of data sent over the wireless network is greatly diminished. The gateway also checks the WML for errors as part of this process, which is rigorous as WML is derived from XML and has strict syntax requirements.

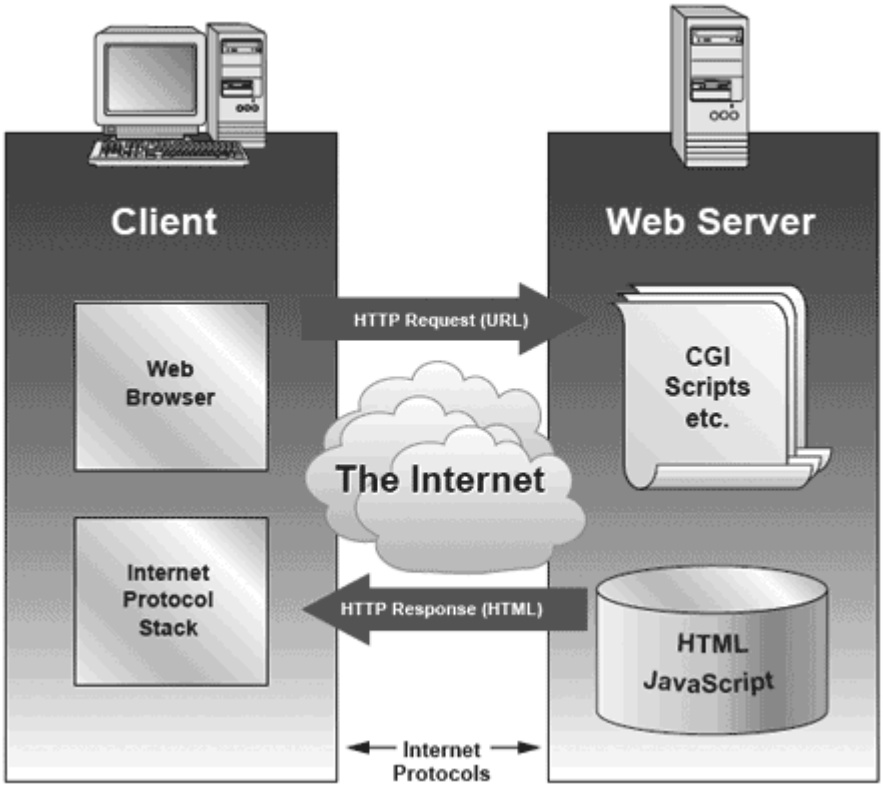
## WEB MODEL VERSUS THE WAP MODEL

### The Internet Model:

The Internet model makes it possible for a client to reach services on a large number of origin servers, each addressed by a unique Uniform Resource Locator (URL).

The content stored on the servers is of various formats, but HTML is the predominant. HTML provides the content developer with a means to describe the appearance of a service in a flat document structure. If more advanced features like procedural logic are needed, then scripting languages such as JavaScript or VB Script may be utilised.

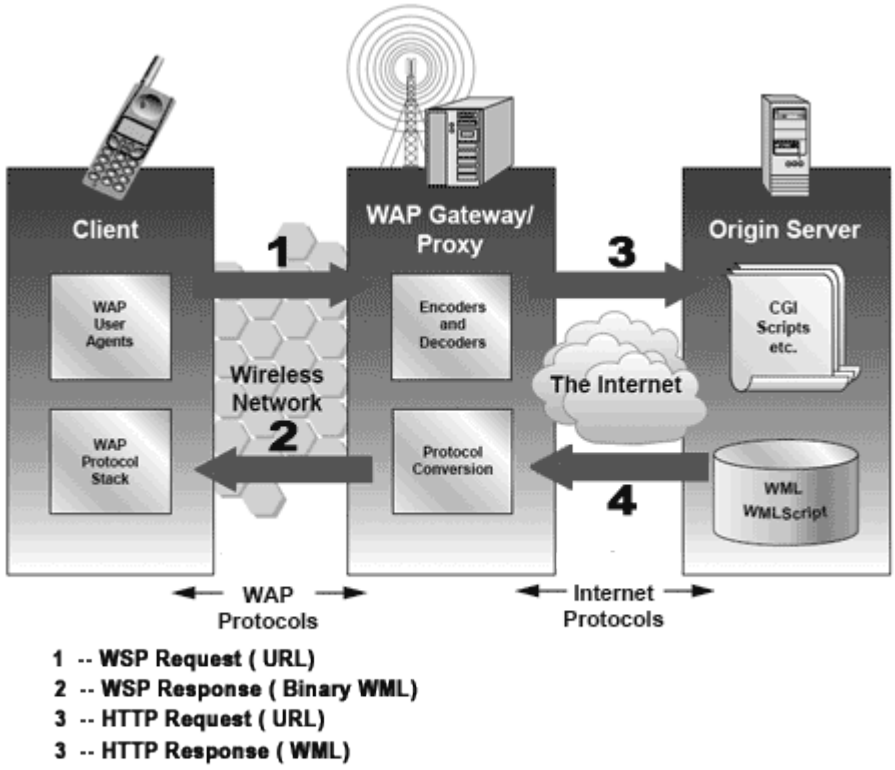
The figure below shows how a WWW client request a resource stored on a web server. On the Internet standard communication protocols, like HTTP and Transmission Control Protocol/Internet Protocol (TCP/IP) are used.




The content available at the web server may be static or dynamic. Static content is produced once and not changed or updated very often; for example, a company presentation. Dynamic content is needed when the information provided by the service changes more often; for example, timetables, news, stock quotes, and account information. Technologies such as Active Server Pages (ASP), Common Gateway Interface (CGI), and Servlets allow content to be generated dynamically.

**The WAP Model:**

The figure below shows the WAP programming model. Note, the similarities with the Internet model. Without the WAP Gateway/Proxy, the two models would have been practically identical.




WAP Gateway/Proxy is the entity that connects the wireless domain with the Internet. You should make a note that the request that is sent from the wireless client to the WAP Gateway/Proxy uses the Wireless Session Protocol (WSP). In its essence, WSP is a binary version of HTTP.

A markup language - the Wireless Markup Language (WML) has been adapted to develop optimized WAP applications. In order to save valuable bandwidth in the wireless network, WML can be encoded into a compact binary format. Encoding WML is one of the tasks performed by the WAP Gateway/Proxy.

**How WAP Model Works?**

When it comes to actual use, WAP works like this:

- The user selects an option on their mobile device that has a URL with Wireless Markup language (WML) content assigned to it.
- The phone sends the URL request via the phone network to a WAP gateway using the binary encoded WAP protocol.
- The gateway translates this WAP request into a conventional HTTP request for the specified URL and sends it on to the Internet.
- The appropriate Web server picks up the HTTP request.
- The server processes the request just as it would any other request. If the URL refers to a static WML file, the server delivers it. If a CGI script is requested, it is processed and the content returned as usual.

- The Web server adds the HTTP header to the WML content and returns it to the gateway.
- The WAP gateway compiles the WML into binary form.
- The gateway then sends the WML response back to the phone.
- The phone receives the WML via the WAP protocol.
- The micro-browser processes the WML and displays the content on the screen.

Web Model WAP Model 1. TCP/IP Protocol is used 1. WAP stack protocol is used between device and gateway , TCP/IP is used between gateway and server 2. HTTP request and response are human readable 2. WSP request and response are encoded into a compact binary form. 3. HTTP proxy can be used 3. WAP gateway is always used. 4. Client- side scripting (java script and VBScript) embedded in the HTML code befor being interpeted by the browser 4. client –side scripting (WMLScript only )is in a separate file, the user – Agent must have an interpreter for byte code. 5. Browser support a large number of images formats 5. Mobile user agent support a smaller number of multimedia formats like WBMP file.

**POSITIONING OF A WAP GATEWAY IN THE NETWORK:**

1. WAP gateway provided by network operator.

2. WAP gateway provided by content provider.

3. WAP gateway provided by ISP (internet service provider).

1.by Network Operator:

Advantage: •

One single gateway installation to get access any internet content.

Disadvantages:

a) The network operator might introduce additional content.

b) Even if secure HTTP (HTTPS) and SSL are used, the requested content will be in an unencrypted form in the main memory (cache) of the WAP Gateway.

c) The network operator may choose to block access to all but a few 'approved' WAP sites.

2.by Content Provider:

Advantage:

• Secure applications.

Disadvantages:

• Mobile user must to have all necessary gateway configurations setup on their devices, for example, Nokia 7110 supports up to 10 gateway configurations, so he must to switch configurations every time he needs to access a different WAP sites.

3.by ISP

• This solution is technically possible; it would hardly make any sense commercially.

Gateway examples

• Ericsson:

o WAP gateway / proxy for GSM networks.

o Jambala WAP gateway for TDMA networks.

• Kannel, open- source WAP gateway (free for unix )

• KNO software, free personal WAP gateway.

• Up.link server.

• Nokia server

**BASIC WML:**

The topmost layer in the WAP (Wireless Application Protocol) architecture is made up of WAE (Wireless Application Environment), which consists of WML and WML scripting language.

- WML stands for **W**ireless **M**arkup **L**anguage
- WML is an application of XML, which is defined in a document-type definition.
- WML is based on HDML and is modified so that it can be compared with HTML.
- WML takes care of the small screen and the low bandwidth of transmission.
- WML is the markup language defined in the WAP specification.
- WAP sites are written in WML, while web sites are written in HTML.
- WML is very similar to HTML. Both of them use tags and are written in plain text format.
- WML files have the extension ".wml". The MIME type of WML is "text/vnd.wap.wml".
- WML supports client-side scripting. The scripting language supported is called WMLScript.

**Wireless Markup Language** (WML), based on XML, is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones. It

provides navigational support, data input, hyperlinks, text and image presentation, and forms, much like HTML (HyperText Markup Language). It preceded the use of other markup languages now used with WAP, such as HTML itself, and XHTML (which are gaining in popularity as processing power in mobile devices increases).

WML Versions:

WAP Forum has released a latest version WAP 2.0. The markup language defined in WAP 2.0 is XHTML Mobile Profile (MP). The WML MP is a subset of the XHTML. A style sheet called WCSS (WAP CSS) has been introduced alongwith XHTML MP. The WCSS is a subset of the CSS2.

Most of the new mobile phone models released are WAP 2.0-enabled. Because WAP 2.0 is backward compatible to WAP 1.x, WAP 2.0-enabled mobile devices can display both XHTML MP and WML documents.

WML 1.x is an earlier technology. However, that does not mean it is of no use, since a lot of wireless devices that only supports WML 1.x are still being used. Latest version of WML is 2.0 and it is created for backward compatibility purposes. So WAP site developers need not to worry about WML 2.0.

**WML Decks and Cards:**

A main difference between HTML and WML is that the basic unit of navigation in HTML is a page, while that in WML is a card. A WML file can contain multiple cards and they form a deck.

When a WML page is accessed from a mobile phone, all the cards in the page are downloaded from the WAP server. So if the user goes to another card of the same deck, the mobile browser does not have to send any requests to the server since the file that contains the deck is already stored in the wireless device.

You can put links, text, images, input fields, option boxes and many other elements in a card.

**WML Program Structure:**

Following is the basic structure of a WML program:

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"

"http://www.wapforum.org/DTD/wml12.dtd">



<wml>



<card id="one" title="First Card">
```

```
<p>

This is the first card in the deck

</p>

</card>



<card id="two" title="Second Card">

<p>

Ths is the second card in the deck

</p>

</card>



</wml>
```

The first line of this text says that this is an XML document and the version is 1.0. The second line selects the document type and gives the URL of the document type definition (DTD).

One WML deck (i.e. page ) can have one or more cards as shown above. We will see complete details on WML document structure in subsequent chapter.

Unlike HTML 4.01 Transitional, text cannot be enclosed directly in the <card>...</card> tag pair. So you need to put a content inside ... as shown above.

To develop WAP applications, you will need the following:

- **A WAP enabled Web Server:** You can enable your Apache or Microsoft IIS to serve all the WAP client request.
- **A WAP Gateway Simulator:** This is required to interact to your WAP server.
- **A WAP Phone Simulator:** This is required to test your WAP Pages and to show all the WAP pages.

You can write your WAP pages using the following languages:

- Wireless Markup Language(WML) to develop WAP application.
- WML Script to enhance the functionality of WAP application.

**A BASIC WML CARD:**

The <card> element supports the following attributes:

| Attribute | Value | Description |
|---|---|---|
| title | cdata | Gives a title to this card. This title is displayed in some way by the browser when the card is visible. |
| newcontext | • true<br>• false | Specifies that when this card is entered, the browser context should be cleared. |
| ordered | • true<br>• false | Provides a hint to the browser about how the card is organized. Set it to true if the card consists of a number of separate fields that should be dealt with in the order they appear in the card. Set it to false if the card contains optional fields or may be filled in out of order. |
| onenterforward | URL | Occurs when the user navigates into a card using a "go" task |
| onenterbackward | URL | Occurs when the user navigates into a card using a "prev" task |
| ontimer | URL | Occurs when a "timer" expires |
| xml:lang | language_code | Sets the language used in the element. |
| class | cdata | Sets a class name for the element. |
| id | element_ID | A unique ID for the element. |

Example:

Following is the example showing usage of this element:

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"

"http://www.wapforum.org/DTD/wml12.dtd">

<wml>
```

```
<card id="one" title="First Card">
<p>
This is the first card in the deck
</p>
</card>

<card id="two" title="Second Card">
<p>
Ths is the second card in the deck
</p>
</card>

</wml>
```

**TEXT FORMATTING:**

These tags can affect the layout of a text:

| | |
|---|---|
| **<b>** | Defines bold text |
| **<big>** | Defines big text |
| **** | Defines emphasized text |
| **<i>** | Defines italic text |
| **<small>** | Defines small text |
| **** | Defines strong text |
| **<u>** | Defines underlined text |

**Attributes**

| **Attribute** | **Value** | **Description** |
|---|---|---|
| xml:lang | *language_code* | Sets the language used in the element |
| class | *cdata* | Sets a class name for the element. The class name is case sensitive. An element can be connected to multiple classes. Multiple class names within the class attribute are separated by white space |

| | | |
|---|---|---|
| id | *id* | Sets a unique name for the element |

**Examples**

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
<card title="Formatting">
<p>
 <em>emphasized text</em><br/>
 <strong>strong text</strong><br/>
 <b>bold text</b><br/>
 <i>italic text</i><br/>
 <u>underlined text</u><br/>
 <big>big text</big><br/>
 <small>small text</small>
</p>
</card>
</wml>
```

**WML Text and Text Formatting**

By this point, you have seen how to create all sorts of interactive applications in WML and how to use variables, tasks, and events to do things that would require serious server-side processing in HTML.

Now that there aren't any exciting new features to cover, it's time to go back to simple text and see what can be done with it to make it more interesting. This is an area where WML is seriously lacking in comparison to HTML, which provides all sorts of features for changing the size, style, typeface, and color of text, as well as powerful support for tables.

One reason for not covering this topic earlier is that it shouldn't be your major concern when creating WML. Most cell phone browsers simply ignore all these text-formatting options, so a large number of users never see all the styles you spend so long choosing. Text style changes can be used to make your WAP content look much more attractive on PDAs and future smart cell phones, but you should not think of them as more than just little decorations to be added on once everything is working properly. Even features such as tables and centered or right-aligned text are not

guaranteed to be present in all browsers. Having said that, the first element we look at is one that's always present in some form—the <p> element.

**The <p> Element**

As you saw back in Example 1.1, the <p> element marks a paragraph of text in a WML card. All body text, user controls (such as <input> and <select> elements), ...

**The <br> Element**

The <br> element is one of the simplest in WML. It takes no attributes and is

always specified as an empty-element tag, <br/>. It marks a line break in a

paragraph of text: when the browser encounters a <br/>, it starts a new line.

You may ask what the difference is between using a <br/> to break lines:

```
<p>text<br/>more text</p>
```

and using another paragraph:

```
<p>text</p><p>more text</p>
```

The answer is that WAP doesn't specify the difference. Some browsers insert a small amount of blank space between paragraphs, but won't do this at a <br/>, but not all browsers actually make a distinction at all. Note that if you need to change the alignment or wrapping mode of the text, you have to use a <p>, since the <br/>tag can't specify these attributes.

A good rule of thumb is to use <p> where the text naturally breaks into blocks

, just like the paragraphs in normal text. Use <br/> where you want a break for presentation, but the text continues with part of the same block afterward. For example, when putting normal paragraphs of text into a deck, use one <p> element for each paragraph:

```
<p>
  A good rule of thumb is to use &lt;p&gt; where the
  text naturally breaks into blocks, just like the
  paragraphs in normal text. Use &lt;br/&gt; where
  you want a break for presentation, but the text
```

```
    continues with part of the same block afterward.
  </p>
  <p>
    For example, when putting normal paragraphs of
    text into a deck, use one &lt;p&gt; element for
    each paragraph:
  </p>
```

**Character Formatting**

The support for character formatting in WML is quite limited compared to HTML. There is no support for specifying the color or typeface of the text, size changes are limited to “bigger” or “smaller,” and there’s no guarantee that any of these choices will be honored anyway.

Support is provided through seven elements. None take any attributes, and their effect applies to all the text they enclose. The browser is free to ignore any or all these attributes if it chooses or if its screen display can’t cope with them.

**Emphasis**

These two elements provide some kind of emphasis. The exact form this takes isn’t specified: some browsers may use style changes such as boldface or italics to emphasize the text, some may change the color, and some may ignore it altogether. These tags are the most likely of the character formats to be supported, and so you should use these in preference to the other formats. If you don’t need a specific style but just want some text to stand out, you should use the following two tags instead of <b>, <i>, and <u>:

```
  <em>
    Emphasis
     <strong>
    Strong emphasis
```

**Size Changes**

These two elements provide simple size changes in the text:

```
  <big>
    Larger font size
```

<small>

Smaller font size

**Style Changes**

These three elements provide simple text-style changes. Because many browsers don't support these elements, you should use these only if you really need exact control over how the text is displayed:

<b>

Boldface

<i>

Italics

<u>

Underlining

**Tables**

Tables are one of the worst-supported features in WML, at least in browsers available at the time of writing. The reason for this is that displaying tables properly (as laid down in the WAP specifications) often requires a lot of screen space, which is at a premium on devices such as cell phones. For example, at least one browser currently available displays each cell of a table on a new line, with lines of * characters to mark the places where rows should have started.

WML also doesn't allow user interface elements to appear in tables, except for anchored text (using the <a> or<anchor> elements). This makes it easier for those browsers that do support tables. You are, however, allowed images, text-style changes, and even line breaks.

WML tables include a number of rows, each containing a number of cells. The cells may themselves contain multiple lines of text, due to embedded <br/> tags, but these are all considered part of the same cell.

**The <table> Element**

This element declares a table in a WML card. It must appear inside a paragraph (in other words, inside a <p>element). A <table> contains nothing but <tr> elements giving its rows. It takes three attributes, giving details about the intended presentation of the table onscreen.

**Attributes of the <table> element**

columns (required number)

Specifies the number of columns in the table. If a row (a <tr> element) has fewer than this number of cells (<td> elements), the browser pads it with empty cells at the end of the row.

...

**Navigation**

WML provides a number of ways to navigate within and between decks of cards. To understand navigation, it's important to examine two related key WML concepts—the "event" and the "task". Basically, events trigger tasks. Events may be user generated, such as a key press or the selection of an on-screen link; events may also be generated by the wireless device itself, such as when a countdown timer expires. A task represents the execution of a piece of code in response to an event. Tasks typically are used to control navigation either within a deck or when loading an entirely new deck.

Events and tasks lead us to the <do> element for navigation. The <do> element provides a way for the user to act upon the currently displayed card. The <do> element must be paired with a task element, such as <go>, in order to perform an action. The <go> task element is used for both inter-card navigation and for making server requests.

**Inter-Card Navigation**

Figure 5 shows how the <do> and <go> elements work together to provide inter-card navigation. Note the use of the # character to designate that the target (MainCategories) is a card within the current deck. This attribute can also be used to jump directly into a specified card in a new deck (e.g.,http://gov.ns.ca/health/inspection.wml#SubCategories). Note, also, the use of comments as defined by XML (<!--*comment text...*-->).

```
<wml>

 <!--First card is splash screen-->

 <card>

  <do type="accept" label="Categories">

   <go href="#MainCategories"/>

  </do>

  <p align="center">
```

```
  ...
  </p>
 </card>
 <!--Second card displays categories-->
 <card id="MainCategories">
  <p mode="nowrap">
  ...
  </p>
 </card>
</wml>
```

***Card Navigation***

**Inter-Deck Navigation**

For navigation to a new deck, WML supports a do-go event handler that references a server URL. Figure 6 illustrates a modified version of the second card from the previous example. Here the href attribute of the go element refers to the URL of a CGI script that will presumably deliver the appropriate WML deck. It's possible to pass data to the specified URL using post fields or by adding parameters. Note themethod attribute, which is used to specify whether the request should be made using a get (data will be included in the request) or a post (data will sent separately). The method attribute is not mandatory, and the default is get.

```
 <card id="MainCategories">
  <do type="accept" label="Details">
   <go method="post" href="http://gov.ns.ca/health/inspect.cgi"/>
  </do>
```

```
<p mode="nowrap">

  ...

</p>

</card>
```

***Deck Navigation***

When there are a number of possible navigation choices from a single page, you can use a series of do-go event handlers, specifying 'type="options"' for each, as shown in Figure 7. When the user selects an option (e.g., Premises), control will pass to the URL referenced by the 'href' attribute (e.g., http://gov.ns.ca/health/inspect.cgi).

```
<!--Second card displays categories-->>

 <card id="MainCategories">

  <do type="options" label="Staff">

   <go method="post" href="http://gov.ns.ca/health/inspect.cgi"/>

  </do>

  <do type="options" label="Food">

   <go method="post" href="http://gov.ns.ca/health/inspect.cgi"/>

  </do>

  <do type="options" label="Premises">

   <go method="post" href="http://gov.ns.ca/health/inspect.cgi"/>
```

```
</do>

<do type="options" label="Animals">

  <go method="post" href="http://gov.ns.ca/health/inspect.cgi"/>

</do>

<p mode="nowrap">

...

</p>

</card>
```

# UNIT III

## INTERACTING WITH THE USER

- Making a Selection
- Events
- Variables
- Input and Parameter Passing

## WML SCRIPT

- Need for WML Script
- Lexical Structure
- Variables and Literals
- Operators
- Automatic Type Conversion
- Control Constructs Functions
- Standard Libraries
- Pragmas
- Dealing with Errors

## Interacting with the User:

The <input/> element is used to create input fields and input fields are used to obtain alphanumeric data from users.

### Attributes:

This element supports the following attributes:

| Attribute | Value | Description |
|---|---|---|
| name | Text | The name of the variable that is set with the result of the user's input |
| maxlength | Number | Sets the maximum number of characters the user can enter in the field |
| emptyok | • true<br>• false | Sets whether the user can leave the input field blank or not. Default is "false" |
| format | A<br>a<br>N<br>X<br>x<br>M<br>m<br>*f<br>*nf* | Sets the data format for the input field. Default is "*M".<br>A = uppercase alphabetic or punctuation characters<br>a = lowercase alphabetic or punctuation characters<br>N = numeric characters<br>X = uppercase characters<br>x = lowercase characters<br>M = all characters<br>m = all characters<br>**f* = Any number of characters. Replace the *f* with one of the letters above to specify what characters the user can enter<br>*nf* = Replace the *n* with a number from 1 to 9 to specify the number of characters the user can enter. Replace the *f* with |

| | | |
|---|---|---|
| | | one of the letters above to specify what characters the user can enter |
| size | Number | Sets the width of the input field |
| tabindex | Number | Sets the tabbing position for the select element |
| title | Text | Sets a title for the list |
| type | • text<br>• password | Indicates the type of the input field. The default value is "text". Password field is used to take password for authentication purpose. |
| value | Text | Sets the default value of the variable in the "name" attribute |
| xml:lang | language_code | Sets the language used in the element |
| class | class data | Sets a class name for the element. |
| id | element ID | A unique ID for the element. |

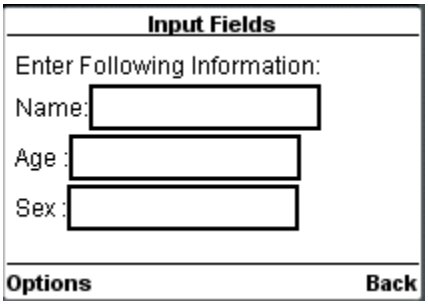
Example:

Following is the example showing usage of this element.

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"

"http://www.wapforum.org/DTD/wml12.dtd">


<wml>


<card title="Input Fields">

<p> Enter Following Information:<br/>
```

```
Name: <input name="name" size="12"/>

Age :  <input name="age" size="12" format="*N"/>

Sex :  <input name="sex" size="12"/>

</p>

</card>


</wml>
```

This will provide you the following screen to enter required information:




## MAKING A SELECTION

The <select>...</select> WML elements are used to define a selection list and the <option>...</option> tags are used to define an item in a selection list. Items are presented as radio buttons in some WAP browsers. The <option>...</option> tag pair should be enclosed within the <select>...</select> tags.

### Attributes:

This element supports the following attributes:

| Attribute | Value | Description |
|---|---|---|
| iname | text | Names the variable that is set with the index result of the selection |

| | | |
|---|---|---|
| ivalue | text | Sets the pre-selected option element |
| multiple | • true<br>• false | Sets whether multiple items can be selected. Default is "false" |
| name | text | Names the variable that is set with the result of the selection |
| tabindex | number | Sets the tabbing position for the select element |
| title | text | Sets a title for the list |
| value | text | Sets the default value of the variable in the "name" attribute |
| xml:lang | language_code | Sets the language used in the element |
| class | class data | Sets a class name for the element. |
| id | element ID | A unique ID for the element. |

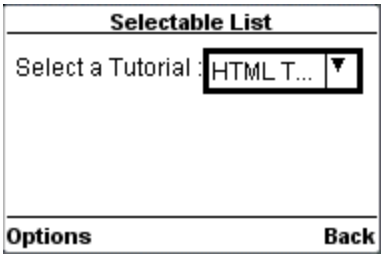
Example:

Following is the example showing usage of this element.

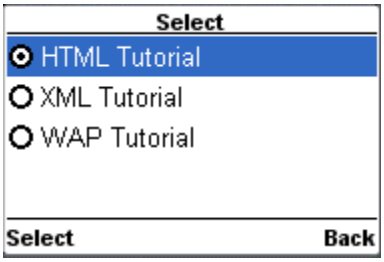
```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"

"http://www.wapforum.org/DTD/wml12.dtd">


<wml>


<card title="Selectable List">

<p> Select a Tutorial :

 <select>
```

```
 <option value="htm">HTML Tutorial</option>
 <option value="xml">XML Tutorial</option>
 <option value="wap">WAP Tutorial</option>
</select>
</p>
</card>


</wml>
```

When you will load this program, it will show you the following screen:




Once you highlight and enter on the options, it will display the following screen:

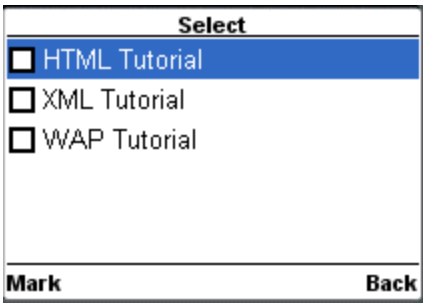



You want to provide option to select multiple options, then set *multiple* attribute to *true* as follows:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">


<wml>
```

```
<card title="Selectable List">
<p> Select a Tutorial :
 <select multiple="true">
  <option value="htm">HTML Tutorial</option>
  <option value="xml">XML Tutorial</option>
  <option value="wap">WAP Tutorial</option>
 </select>
</p>
</card>

</wml>
```

This will give you a screen to select multiple options as follows:




## WML – Events

Event in ordinary language can be defined as something happened. In programming, **event**is identical in meaning, but with one major difference. When something happens in a computer system, the system itself has to **(1)** detect that something has happened and **(2)**know what to do about it.

WML language also supports events and you can specify an action to be taken whenever an event occurs. This action could be in terms of WMLScript or simply in terms of WML.

WML supports following four event types:

- onenterbackward: This event occurs when the user hits a card by normal backward navigational means. That is, user presses the Back key on a later card and arrives back at this card in the history stack.
- onenterforward: This event occurs when the user hits a card by normal forward navigational means.
- onpick: This is more like an attribute but it is being used like an event. This event occurs when an item of a selection list is selected or deselected.
- ontimer: This event is used to trigger an event after a given time period.

These event names are case sensitive and they must be lowercase.

### WML <onevent> Element:

The <onevent>...</onevent> tags are used to create event handlers. Its usage takes the following form:

```
<onevent type="event_type">
  A task to be performed.
</onevent>
```

You can use either *go, prev* or *refresh* task inside <onevent>...</onevent> tags against an event.

The <onevent> element supports the following attributes:

| Attribute | Value | Description |
|---|---|---|
| type | • onenterbackward<br>• onenterforward<br>• onpick<br>• ontimer | Defines a type of event occured. |

| | | |
|---|---|---|
| class | class data | Sets a class name for the element. |
| id | element ID | A unique ID for the element. |

Following is the example showing usage of <onevent> element. In this example, whenever you try to go back from second card to first card then **onenterbackward** occurs which moves you to card number three. Copy and paste this program and try to play with it.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">

<wml>
<onevent type="onenterbackward">
 <go href="#card3"/>
</onevent>

<card id="card1" title="Card 1">
<p>
 <anchor>
   <go href="#card2"/>
   Go to card 2
 </anchor>
</p>
</card>
<card id="card2" title="Card 2">
<p>
```

```
  <anchor>
  <prev/>
    Going backwards
  </anchor>
</p>
</card>
<card id="card3" title="Card 3">
<p>
Hello World!
</p>
</card>
</wml>
```

## WML - Variables

Because multiple cards can be contained within one deck, some mechanism needs to be in place to hold data as the user traverses from card to card. This mechanism is provided via WML variables.

WML is case sensitive. No case folding is performed when parsing a WML deck. All enumerated attribute values are case sensitive. For example, the following attribute values are all different: id="Card1", id="card1", and id="CARD1".

Variables can be created and set using several different methods. Following are two examples:

## The <setvar> element:

The <setvar> element is used as a result of the user executing some task. The >setvar> element can be used to set a variable's state within the following elements: <go>, <prev>, and <refresh>.

This element supports the following attributes:

| Attribute | Value | Description |
|---|---|---|
| name | String | Sets the name of the variable |
| value | String | Sets the value of the variable |
| class | class data | Sets a class name for the element. |
| id | element ID | A unique ID for the element. |

The following element would create a variable named *a* with a value of 1000:

```
<setvar name="a" value="1000"/>
```

**The input elements:**

Variables are also set through any input element like *input,select, option,* etc. A variable is automatically created that corresponds with the named attribute of an input element.

For example, the following element would create a variable named *b*:

```
<select name="b">
<option value="value1">Option 1</option>
<option value="value2">Option 2</option>
</select>
```

**Using Variables:**

Variable expansion occurs at runtime, in the microbrowser or emulator. This means it can be concatenated with or embedded in other text.

Variables are referenced with a preceding dollar sign, and any single dollar sign in your WML deck is interpreted as a variable reference.

```
<p> Selected o
```

## INPUT AND PARAMETER PASSING

Many times, you will want your users to submit some data to your server. Similar to *HTML Form* WML also provide a mechanism to submit user data to web server.

To submit data to the server in WML, you need the <go>...</go> along with <postfield/> tags. The <postfield/> tag should be enclosed in the <go>...</go> tag pair.

To submit data to a server, we collect all the set WML variables and use <postfield> elements to send them to the server. The <go>...</go> elements are used to set posting method to either POST or GET and to specify a server side script to handle uploaded data.

In previous chapters we have explained various ways of taking inputs form the users. These input elements sets WML variables to the entered values. We also know how to take values from WML variables. So now following example shows how to submit three fields *name, age* and *sex* to the server.

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"

"http://www.wapforum.org/DTD/wml12.dtd">



<wml>



<card id="card1" title="WML Form">

<p>

  Name: <input name="name" size="12"/>

  Sex : <select name="sex">

    <option value="male">Male</option>
```

```
    <option value="female">Female</option>
    </select>
  Age :  <input name="age" size="12" format="*N"/>
  <anchor>
    <go method="get" href="process.php">
      <postfield name="name" value="$(name)"/>
      <postfield name="age" value="$(age)"/>
      <postfield name="sex" value="$(sex)"/>
    </go>
    Submit Data
  </anchor>
</p>
</card>


</wml>
```

When you download above code on your WAP device, it will provide you option to enter three fields *name, age* and *sex* and one link *Submit Data*. You will enter three fields and then finally you will select *Submit Data* link to send entered data to the server.

The *method* attribute of the <go> tag specifies which HTTP method should be used to send the form data.

If the HTTP POST method is used, the form data to be sent will be placed in the message body of the request. If the HTTP GET method is used, the form data to be sent will be appended to the URL. Since a URL can only contain a limited number of characters, the GET method has the disadvantage that there is a size limit for the data to be sent. If the user data contains non-ASCII characters, you should make use of the POST method to avoid encoding problems.

There is one major difference between HTML and WML. In HTML, the name attribute of the <input> and <select> tags is used to specify the name of the parameter to be sent, while in WML the name attribute of the <postfield> tag is used to do the same thing. In WML, the name attribute of <input> and <select> is used to specify the name of the variable for storing the form data.

If you already know how to write server side scripts for Web Application, then for you this is very simple to write Server Side program for WML applications. You can use your favorite server-side technology to do the processing required by your mobile Internet application.

At the server side, the parameter name will be used to retrieve the form data.

Consider the following example from previous chapter to submit name, age and sex of a person:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">

<wml>

<card id="card1" title="WML Form">
<p>
  Name: <input name="name" size="12"/>
  Sex : <select name="sex">
    <option value="male">Male</option>
    <option value="female">Female</option>
    </select>
  Age :  <input name="age" size="12" format="*N"/>
  <anchor>
```

```
    <go method="get" href="process.php">
      <postfield name="name" value="$(name)"/>
      <postfield name="age" value="$(age)"/>
      <postfield name="sex" value="$(sex)"/>
    </go>
    Submit Data
  </anchor>
</p>
</card>


</wml>
```

**WML SCRIPT: Need for WML script**

**WMLScript** is a procedural programming language and dialect of JavaScript used for WML pages and is part of the Wireless Application Protocol (WAP).

WMLScript is a client-side scripting language and is similar to JavaScript. Just like JavaScript WMLScript is used for tasks such as user input validation, generation of error message and other Dialog boxes etc.

WMLScript is based on ECMAScript (European Computer Manufacturers Association Script), which is JavaScript's standardized version. Thus the syntax of WMLScript is similar to JavaScript but not fully compatible.[1]

Despite the syntactical similarities, they are two different languages. WMLScript does not have objects or array, which JavaScript has. On the other hand, it allows you to declare and include external functions from other scripts. WMLScript is optimised for low power devices, and is a compiled language.

WMLScript (Wireless Markup Language Script) is the client-side scripting language of WML (Wireless Markup Language). A scripting language is similar to a programming language, but is of lighter weight.

With WMLScript, the wireless device can do some of the processing and computation. This reduces the number of requests and responses to/from the server.

This chapter will give brief description of all the important WML Script components.

## WML Script Components:

WML Script is very similar to Java Script. Almost WML Script components have similar meaning as they have in Java Script. A WML Script program components are summarized as follows:

## WML Script Operators:

WML Script supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Check for complete detail of **The WML Operators**.

## WML Script Control Statements:

Control statements are used for controlling the sequence and iterations in a program.

| Statement | Description |
|---|---|
| if-else | Conditional branching |
| for | Making self-incremented fixed iteration loop |
| while | Making variable iteration loop |

| | |
|---|---|
| break | Terminates a loop |
| continue | Quit the current iteration of a loop |

Check for complete detail of **WML Script Control Statements**.

**WML Script Functions:**

The user-defined functions are declared in a separate file having the extension .wmls. Functions are declared as follows:

```
function name (parameters)
{
  control statements;
  return var;
}
```

The functions used are stored in a separate file with the extension .wmls. The functions are called as the filename followed by a hash, followed by the function name:

```
maths.wmls#squar()
```

**WML Scripts Standard Libraries:**

There are six standard libraries totally. Here is an overview of them:

- **Lang:** The Lang library provides functions related to the WMLScript language core.

  **Example Function:** abs(),abort(), characterSet(),float(), isFloat(), isInt(), max(), isMax(), min(), minInt(), maxInt(), parseFloat(), parseInt(), random(), seed()

- **Float:** The Float library contains functions that help us perform floating-point arithmetic operations.

  **Example Function:** sqrt(), round(), pow(), ceil(), floor(), int(), maxFloat(), minFloat()

- **String:** The String library provides a number of functions that help us manipulate strings.

  **Example Function:** length(), charAt(), find(), replace(), trim(), compare(), format(), isEmpty(), squeeze(), toString(), elementAt(), elements(), insertAt(), removeAt(), replaceAt()

- **URL:** The URL library contains functions that help us manipulate URLs.

  **Example Function:** getPath(), getReferer(), getHost(), getBase(), escapeString(), isValid(), loadString(), resolve(), unescapeString(), getFragment()

- **WMLBrowser:** The WMLBrowser library provides a group of functions to control the WML browser or to get information from it.

  **Example Function:** go(), prev(), next(), getCurrentCard(), refresh(), getVar(), setVar()

- **Dialogs:** The Dialogs library Contains the user interface functions.

  **Example Function:** prompt(), confirm(), alert()

**WML Scripts Comments:**

There are two types of comments in WMLScript:

- **Single-line comment:** To add a single-line comment, begin a line of text with the // characters.
- **Multi-line comment:** To add a multi-line comment, enclose the text within /* and */.

These rules are the same in WML Script, JavaScript, Java, and C++. The WML Script engine will ignore all comments. The following WML Script example demonstrates the use of comments:

```
// This is a single-line comment.

 /* This is a
   multi-line comment. */
```

```
/* A multi-line comment can be placed on a single line. */
```

**WML Script Case Sensitivity:**

The WML Script language is case-sensitive. For example, a WML Script function with the name WML Script Function is different from wml script function. So, be careful of the capitalization when defining or referring to a function or a variable in WML Script.

**Whitespaces in WML Script:**

Except in string literals, WML Script ignores extra whitespaces like spaces, tabs and newlines. Hence, the code in the earlier "Hello World" example can be typed in the following way and the result will remain the same:

**WML Script Statement Termination by Semicolons:**

A semicolon is required to end a statement in WML Script. This is the same as C++ and Java. Note that JavaScript does not have such requirement but WML Script makes it mandatory.

## 5.1 Why Scripting?

WMLScript is designed to provide general scripting capabilities to the WAP architecture. Specifically, WMLScript can be used to complement the Wireless Markup Language [WML]. WML is a markup language based on Extensible Markup Language [XML]. It is designed to be used to specify application content for narrowband devices like cellular phones and pagers. This content can be represented with text, images, selection lists etc. Simple formatting can be used to make the user interfaces more readable as long as the client device used to display the content can support it.
However, all this content is *static* and there is no way to extend the language without modifying WML itself. The following list contains some capabilities that are not supported by WML:

- Check the validity of user input (validity checks for the user input)
- Access to facilities of the device. For example, on a phone, allow the programmer to make phone calls, send messages, add phone numbers to the address book, access the SIM card etc.
- Generate messages and dialogs locally thus reducing the need for expensive round-trip to show alerts, error messages, confirmations etc.
- Allow extensions to the device software and configuring a device after it has been deployed.

WMLScript was designed to overcome these limitations and to provide programmable functionality that can be used over narrowband communication links in clients with limited capabilities.

## 5.2 Benefits of using WMLScript

Many of the services that can be used with thin mobile clients can be implemented with WML. Scripting enhances the standard browsing and presentation facilities of WML with behavioural capabilities. They can be used to supports more advanced UI functions, add intelligence to the client, provide access to the device and its peripheral functionality and reduces the amount of bandwidth needed to send data between the server and the client.

WMLScript is loosely based on ECMAScript [ECMA262] and does not require the developers to learn new concepts to be able to generate advanced mobile services.

## 1. WMLSCRIPT CORE

One objective for the WMLScript language is to be close to the core of the ECMAScript Language specification [ECMA262]. The part in the ECMAScript Language specification that defines basic types, variables, expressions and statements is called *core* and can almost be used "as is" for the WMLScript specification. This section gives an overview of the core parts of WMLScript.

See section *WMLScript Grammar* (7) for syntax conventions and precise language grammar.

### Lexical Structure

This section describes the set of elementary rules that specify how you write programs in WMLScript.

#### Case Sensitivity

WMLScript is a case-sensitive language. All language keywords, variables and function names must use the proper capitalisation of letters.

#### Whitespace and Line Breaks

WMLScript ignores spaces, tabs, newlines etc. that appear between tokens in programs, except those that are part of string constants.

**Syntax:**

*WhiteSpace* **::**

*<TAB>*

*<VT>*

*<FF>*

*<SP>*

*<LF>*

*<CR>*

*LineTerminator* **::**

*<LF>*

*<CR>*

*<CR><LF>*

Usage of Semicolons

The following statements in WMLScript have to be followed by a semicolon:**1**

- Empty statement (see section 6.5.1)
- Expression statement (see section 6.5.2)
- Variable statement (see section 6.5.4)
- Break statement (see section 6.5.8)
- Continue statement (see section 6.5.9)

**1** Compatibility note: ECMAScript supports optional semicolons.

- Return statement (see section 6.5.10)

## Comments

The language defines two comment constructs: *line comments* (ie, start with // and end in the end of the line) and *block comments* (ie, consisting of multiple lines starting with /* and ending with */). It is illegal to have nested block comments.**2**

### Syntax:

*Comment* **::**

*MultiLineCom*
*ment*

*SingleLineCo*
*mment*

*MultiLineComment* **::**

**/*** *MultiLineCommentChars*$_{opt}$ ***/**

*SingleLineComment* **::**

**//** *SingleLineCommentChars*$_{opt}$

## Literals

### Integer Literals

Integer literals can be represented in three different ways: decimal, octal and hexadecimal integers.

**Syntax:**

*DecimalIntegerLiteral* **::**

**0**
*NonZeroDigit DecimalDigits*$_{opt}$

*NonZeroDigit* **:: one of**

**1 2 3 4 5 6 7 8 9**

*DecimalDigits* **::**

*DecimalDigit*

*DecimalDigits DecimalDigit*

*DecimalDigit* **:: one of**

**0 1 2 3 4 5 6 7 8 9**

*HexIntegerLiteral* **::**

**0x***HexDigit*

**0X** *HexDigit*
*HexIntegerLiteral*
*HexDigit*

*HexDigit* **:: one of**

**0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F**

---

*OctalIntegerLiteral* **::**

**0** *OctalDigit*
*OctalIntegerLiteral*
*OctalDigit*

*OctalDigit* **:: one of**

**0 1 2 3 4 5 6 7**

The minimum and maximum sizes for integer literals and values are specified in the section 6.2.7.1. An integer literal that is not within the specified value range must result in a compile time error.

### Floating-Point Literals

Floating-point literals can contain a decimal point as well as an exponent.

**Syntax:**

*DecimalFloatLiteral* **::**

*DecimalIntegerLiteral* **.***DecimalDigits*$_{opt}$ *ExponentPart*$_{opt}$

**.** *DecimalDigits* *ExponentPart*$_{opt}$
*DecimalIntegerLiteral ExponentPart*

*DecimalDigits* **::**

*DecimalDigit*

*DecimalDigits DecimalDigit*

*ExponentPart* **::**

*ExponentIndicator SignedInteger*

*ExponentIndicator* **:: one of**

**e E**

*SignedInteger* **::**

*DecimalDigits*

**+***DecimalDigits*

**-***DecimalDigits*

The minimum and maximum sizes for floating-point literals and values are specified in the section
A floating-point literal that is not within the specified value range must result in a compile time error. A floating-point literal underflow results in a floating-point literal zero (0.0).

### String Literals

Strings are any sequence of zero or more characters enclosed within double (") or single quotes (').

**Syntax:**

*StringLiteral* **::**

**"**
*DoubleStringCharacters*$_{opt}$ **"**
**'**
*SingleStringCharacters*$_{opt}$
**'**

Examples of valid strings are:

"Example" 'Specials: \x00 \' \b' "Quote: \""

Since some characters are not representable within strings, WMLScript supports special escape sequences by which these characters can be represented:

| **Sequence** | **Character represented**[3] | **Unicode** | **Symbol** |
|---|---|---|---|
| \’ | Apostrophe or single quote | \u0027 | ' |

| | | | |
|---|---|---|---|
| \" | Double quote | \u0022 | " |
| \\ | Backslash | \u005C | \ |
| \/ | Slash | \u002F | / |
| \b | Backspace | \u0008 | |
| \f | Form feed | \u000C | |
| \n | Newline | \u000A | |
| \r | Carriage return | \u000D | |
| \t | Horizontal tab | \u0009 | |
| **\x***hh* | The character with the encoding specified by two hexadecimal digits *hh* (Latin-1 ISO8859-1) | | |
| *ooo* | The character with the encoding specified by the three octal digits *ooo* (Latin-1 ISO8859-1) | | |
| **\u***hhhh* | The Unicode character with the encoding specified by the four hexadecimal digits *hhhh*. | | |

An escape sequence occurring within a string literal always contributes a character to the string value of the literal and is never interpreted as a line terminator or as a quote mark that might terminate the string literal.

### Boolean Literals

A "truth value" in WMLScript is represented by a boolean literal. The two boolean literals are: true
and false.

**Syntax:**

*BooleanLiteral* **::**
**true**
**false**

### Invalid Literal

WMLScript supports a special *invalid* literal to denote an invalid value.

---

[2] Compatibility note: ECMAScript supports also non-escape characters preceded by a

backslash.

**Syntax:**

*InvalidLiteral* **::**

**invalid**

### Identifiers

Identifiers are used to name and refer to three different elements of WMLScript: variables (see section 6.2), functions (see section 6.4) and pragmas (see section 6.7). Identifiers[4] cannot start with a digit but can start with an underscore (_).

**Syntax:**

*Identifier* **::**

*IdentifierName* **but not** *ReservedWord*

*IdentifierName* **::**

*IdentifierLetter*

*IdentifierName*
*IdentifierLetter*
*IdentifierName*
*DecimalDigit*

*IdentifierLetter* **:: one of**

**a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**

**_**

*DecimalDigit* **:: one of**

**0 1 2 3 4 5 6 7 8 9**

Examples of legal identifiers are:

timeOfDay speed quality HOME_ADDRESS var0 _myName _____

The compiler looks for the longest string of characters make up a valid identifier.

Identifiers cannot contain any special characters except underscore (_). WMLScript keywords and reserved words cannot be used as identifiers. Examples of illegal identifiers are:

while for if my~name $sys 123 3pieces take.this

Uppercase and lowercase letters are distinct which means that the identifiers speed and Speed are different.

### Reserved Words

WMLScript specifies a set of reserved words that have a special meaning in programs and they cannot be used as identifiers. Examples of such words are (full list can be found from the WMLScript grammar specification, see section 7):

---

[3] Compatibility note: ECMAScript supports the usage of $ character in any position of the name, too.

```
break    continue    false    true    while
```

### Name Spaces

WMLScript supports name spaces for identifiers that are used for different purposes. The following name spaces are supported:

- Function names (see section 6.4.1)
- Function parameters (see section 6.4.2) and variables (see section 6.2)
- Pragmas (see section 6.7)

Thus, the same identifiers can be used to specify a function name, variable/parameter name or a name for a pragma within the same compilation unit:

```
use url myTest "http://www.host.com/script";

function myTest(myTest) {
   var value = myTest#myTest(myTest);
   return value;
};
```

## Variables and Data Types

This section describes the two important concepts of WMLScript language: variables and internal data types. A variable is a name associated with a data value. Variables can be used to store and manipulate program data. WMLScript supports local variables[5] only declared inside functions or passed as function parameters.

### Variable Declaration

Variable declaration is compulsory[6] in WMLScript. Variable declaration is done simply by using the
*var* keyword and a variable name (see section 6.5.4 for information about variable statements). Variable names follow the syntax defined for all identifiers.

```
var x;
var   price;
var x,y;
var size = 3;
```

Variables must be declared before they can be used. Initialisation of variables is optional. Uninitialised variables are automatically initialised to contain an empty string ("").

### Variable Scope and Lifetime

The scope of WMLScript variables is the remainder of the function (see section 6.4) in which they have been declared. All variable names within a function must be unique. Block statements (see section 6.5.3) are not used for scoping.

---

[4] Compatibility note: ECMAScript supports global variables, too.
[5] Compatibility note: ECMAScript supports automatic declaration, too.

```
function  priceCheck(givenPrice)  {  if
   (givenPrice > 100) {
      var newPrice = givenPrice;
   } else {
      newPrice = 100;
   };
   return newPrice;
};
```

The lifetime of a variable is the time between the variable declaration and the end of the function.

```
function foo() {
   x = 1;                    // Error: usage before declaration var
   x,y;
```

```
    if (x) {
        var y;              // Error: redeclaration
    };
};
```

### 6.1.2 Variable Access

Variables are accessible only within the function in which they have been declared. Accessing the content of a variable is done by using the variable name:

```
var         myAge

=       37;       var
yourAge =  63;
var ourAge        = myAge + yourAge;
```

## Variable Type

WMLScript is a weakly typed language. The variables are not typed but internally the following basic data types are supported: *boolean*, *integer*, *floating-point* and *string*. In addition to these, a fifth data type *invalid* is specified to be used in cases an invalid data type is needed to separate it from the other internal data types. Since these data types are supported only internally, the programmer does not have to specify variable types and any variable can contain any type of data at any given time.
WMLScript will attempt automatically convert between the different types as needed.

```
var   flag   var        = true;          // Boolean
number                  = 12;            // Integer
var temperature         = 37.7;          // Float
number                  = "XII";         // String
var except              = invalid;       // Invalid
```

### 6.1.3 L-Values

Some operators (see section 6.3.1 for more information about assignment operators) require that the left operand is a reference to a variable (L-value) and not the variable value. Thus, in addition to the five data types supported by WMLScript, a sixth type *variable* is used to specify that a variable name must be provided.

```
result += 111;          // += operator requires a variable
```

## Type Equivalency

WMLScript supports operations on different data types. All operators (see section 6.3) specify the accepted data types for their operands. Automatic data type conversions (see section 6.8) are used to convert operand values to required data types.

## Numeric Values

WMLScript supports two different numeric variable values: *integer* and *floating-point* values[7]. Variables can be initialised with integer and floating-point literals and several operators can be used to modify their values during the run-time. Conversion rules between integer and floating-point values are specified in chapter 6.8

```
var pi          =
3.14; var length =
0; var radius   =
2.5;
length          = 2*pi*radius;
```

## Integer Size

The size of the integer is 32 bits (two's complement). This means that the supported value range[8] for integer values is: -2147483648 and 2147483647. *Lang* [WMLSLibs] library functions can be used to get these values during the run-time:

| | |
|---|---|
| **Lang.maxInt()** | Maximum representable integer value |
| **Lang.minInt()** | Minimum representable integer value |

## Floating-point Size

The minimum/maximum values[9] and precision for floating-point values are specified by [IEEE754]. WMLScript supports 32-bit single precision floating-point format:

- Maximum value: 3.40282347E+38
- Minimum positive nonzero value (at least the normalised precision must be supported): 1.17549435E-38 or smaller

The *Float* [WMLSLibs] library can be used to get these values during the run-time:

| | |
|---|---|
| `Float.maxFloat()` | Maximum representable floating-point value supported. |

[6] Convention: In cases where the value can be either an integer or a floating-point, a more generic term *number* is used instead.

[7] Compatibility note: ECMAScript does not specify maximum and minimum values for integers. All numbers are represented as floating- point values.

[8] Compatibility note: ECMAScript uses double-precision 64-bit format [IEEE754] floating-point values for all numbers.

| | |
|---|---|
| `Float.minFloat()` | Smallest positive nonzero floating-point value supported. |

The special floating-point number types are handled by using the following rules:

- If an operation results in a floating-point number that is not part of the set of finite real numbers (not a number, positive infinity etc.) supported by the single precision floating-point format then the result is an invalidvalue.
- If an operation results in a floating-point *underflow* the result is zero (0.0).
- *Negative* and *positive zero* are equal and undistinguishable.

### String Values

WMLScript supports *strings* that can contain letters, digits, special characters etc. Variables can be initialised with string literals and string values can be manipulated both with WMLScript operators and functions specified in the standard *String* library [WMLSLibs].

```
var msg = "Hello";
var len = String.length(msg);
msg = msg + ' Worlds!';
```

### Boolean Values

*Boolean* values can be used to initialise or assign a value to a variable or in statements which require a boolean value as one of the parameters. Boolean value can be a literal or the result of a logical expression evaluation (see section 6.3.3 for more information).

```
var truth = true;
var lie =
!truth;
```

## Operators and Expressions

The following sections describe the operators supported by WMLScript and how they can be used to form complex expressions.

### Assignment Operators

WMLScript supports several ways to assign a value to a variable. The simplest one is the regular assignment (=) but assignments with operation are also supported:

| Operator | Operation |
|---|---|
| = | Assign |
| += | add (numbers)/concatenate (strings) and assign |
| -= | subtract and assign |
| *= | multiply and assign |
| /= | divide and assign |
| div= | divide (integer division) and assign |

| Operator | Operation |
|---|---|
| %= | remainder (the sign of the result equals the sign of the dividend) and assign |
| <<= | bitwise left shift and assign |
| >>= | bitwise right shift with sign and assign |
| >>>= | bitwise right shift zero fill and assign |
| &= | bitwise AND and assign |
| ^= | bitwise XOR and assign |
| \|= | bitwise OR and assign |

Assignment does not necessarily imply sharing of structure nor does assignment of one variable change the binding of any other variable.

```
var a =
"abc"; var b
=

a;
b = "def";  // Value of a is "abc"
```

## Arithmetic Operators

WMLScript supports all the basic binary arithmetic operations:

| Operator | Operation |
|---|---|
| + | add (numbers)/concatenation (strings) |
| - | subtract |
| * | multiply |
| / | divide |
| div | integer division |

In addition to these, a set of more complex binary operations are supported, too:

| Operator | Operation |
|---|---|
| % | remainder, the sign of the result equals the sign of the dividend |
| << | bitwise left shift |
| >> | bitwise right shift with sign |
| >>> | bitwise shift right with zero fill |
| & | bitwise AND |
| \| | bitwise OR |
| ^ | bitwise XOR |

The basic unary operations supported are:

| Operator | Operation |
|---|---|
| + | plus |
| - | minus |
| -- | pre-or-post decrement |

| Operator | Operation |
|---|---|
| ++ | pre-or-post increment |
| ~ | bitwise NOT |

**Examples:**

```
var y = 1/3;
var x = y*3+(++b);
```

## Logical Operators

WMLScript supports the basic logical operations:

| Operator | Operation |
|---|---|
| && | logical AND |
| \|\| | logical OR |
| ! | logical NOT (unary) |

Logical AND operator evaluates the first operand and tests the result. If the result is false, the result of the operation is false and the second operand is not evaluated. If the first operand evaluates to true, the result of the operation is the result of the evaluation of the second operand. If the first operand evaluates to invalid, the second operand is not

evaluated and the result of the operation is invalid.

Similarly, the logical OR evaluates the first operand and tests the result. If the result is true, the result of the operation is true and the second operand is not evaluated. If the first operand evaluates to false, the result of the operation is the result of the evaluation of the second operand. If the first operand evaluates to invalid, the second operand is not evaluated and the result of the operation is invalid.

```
weAgree = (iAmRight && youAreRight) ||
          (!iAmRight && !youAreRight);
```

WMLScript requires a value of boolean type for logical operations. Automatic conversions from other types to boolean type and *vice versa* are supported (see section 6.8).

**Notice**: If the value of the first operand for logical AND or OR is invalid, the second operand is not evaluated and the result of the operand is invalid:

```
var a = (1/0) || foo(); // result: invalid, no call to foo() var b = true ||
(1/0); // true
var c = false || (1/0); // invalid
```

### 6.1.4 String Operators

WMLScript supports string concatenation as a built-in operation. The + and += operators used with strings perform a concatenation on the strings. Other string operations[10] are supported by a standard *String* library (see [WMLSLibs]).

```
var str = "Beginning" + "End";
var chr = String.charAt(str,10); // chr = "E"
```

### 6.1.5 Comparison Operators

WMLScript supports all the basic comparison operations:

| Operator | Operation |
|---|---|
| < | less than |
| <= | less than or equal |
| == | equal |
| >= | greater or equal |
| > | greater than |
| != | inequality |

Comparison operators use the following rules:

- *Boolean*: trueis larger than false

- *Integer*: Comparison is based on the given integer values
- *Floating-point*: Comparison is based on the given floating-point values
- *String*: Comparison is based on the order of character codes of the given string values. Character codes are defined by the character set supported by the WMLScript Interpreter
- *Invalid*: If at least one of the operands is invalid then the result of the comparison is Invalid

Examples:

```
var res = (myAmount > yourAmount);
var val = ((1/0)  == invalid);      // val = invalid
```

### 6.1.6 Array Operators

WMLScript does not support arrays[11] as such. However, the standard *String* library (see [WMLSLibs]) supports functions by which array like behaviour can be implemented by using strings. A string can contain elements that are separated by a separator specified by the application programmer. For this purpose, the *String* library contains functions by which creation and management of string arrays can be done.

---

[9] Compatibility note: ECMAScript supports String objects and a length attribute for each string. WMLScript does not support objects. However, similar functionality is provided by WMLScript libraries.

[10] Compatibility note: ECMAScript supports arrays.

```
function dummy() {
   var str       = "Mary had a little lamb"; var
   word = String.elementAt(str,4,"  ");
};
```

### 6.1.7 Comma Operator

WMLScript supports the comma (,) operator by which multiple evaluations can be combined into one expression. The result of the comma operator is the value of the second operand:

```
for (a=1, b=100; a < 10; a++,b++) {
   … do something …
};
```

Commas used in the function call to separate parameters and in the variable declarations to separate multiple variable declarations are not comma operators. In these cases, the comma operator must be placed inside the parenthesis:

```
var a=2;
var b=3, c=(a,3);
myFunction("Name", 3*(b*a,c)); // Two parameters:  "Name",9
```

### 6.1.8 Conditional Operator

WMLScript supports the conditional (**?:**) operator which takes three operands. The operator selectively evaluates one of the given two operands based on the boolean value of the first operand. If the value of the first operand (condition) is true then the result of the operation is the result of the evaluation of the second operand. If the value of the first operand is false or invalid then the result of the operation is the result of the evaluation of the third operand.

```
myResult = flag ? "Off" : "On (value=" + level +  ")";
```

**Notice**: This operator behaves like an *if* statement (see section 6.5.5). The third operand is evaluated if the evaluation of the condition results in falseor invalid.

### 6.1.9 typeof Operator

Although WMLScript is a weakly typed language, internally the following basic data types are supported: *boolean*, *integer*, *floating-point*, *string* and *invalid*. Typeof (*typeof*) operator returns an integer value[12] that describes the type of the given expression. The possible results are:

| Type | Code |
|---|---|
| **Integer:** | 0 |
| **Floating-point:** | 1 |
| **String:** | 2 |

[11] Compatibility note: ECMAScript specifies that the *typeof* operator returns a string representing the variable type.

| Type | Code |
|---|---|
| **Boolean:** | 3 |
| **Invalid:** | 4 |

Typeof operator does not try to convert the result from one type to another but returns

the type as it is after the evaluation of the expression.

```
var str           = "123";
var myType = typeof str; // myType = 2
```

### 6.1.10 isvalid Operator

This operator can be used to check the type of the given expression. It returns a boolean value false if the type of the expression is invalid, otherwise true is returned. *isvalid* operator does not try to convert the result from one type to another but returns the type as it is after the evaluation of the expression.

```
var str = "123";
var ok      = isvalid str;              //
true var tst = isvalid (1/0); //  false
```

### 6.1.11 Expressions

WMLScript supports most of the expressions supported by other programming languages. The simplest expressions are constants and variable names, which simply evaluate to either the value of the constant or the variable.

```
567
66.77
"This is too simple"
'This works too'

true
myAccoun
t
```

Expressions that are more complex can be defined by using simple expressions together with operators and function calls.

```
myAccount
+ 3 (a + b)/3
initialValue + nextValue(myValues);
```

### 6.3.12 Expression Bindings

The following table contains all operators supported by WMLScript. The table also contains information about operator precedence (the order of evaluation) and the operator associativity (left-to- right (L) or right-to-left (R)):

| Precedence 13 | Associativity | Operator | Operand types | Result type | Operation performed |
|---|---|---|---|---|---|
| 1 | R | ++ | number | number* | pre- or post-increment (unary) |
| 1 | R | -- | number | number* | pre- or post-decrement (unary) |
| 1 | R | + | number | number* | unary plus |
| 1 | R | - | number | number* | unary minus (negation) |
| 1 | R | ~ | integer | integer* | bitwise NOT (unary) |
| 1 | R | ! | boolean | boolean* | logical NOT (unary) |
| 1 | R | typeof | any | integer | return internal data type (unary) |
| 1 | R | isvalid | any | boolean | check for validity (unary) |
| 2 | L | * | numbers | number* | multiplication |
| 2 | L | / | numbers | floating-point* | division |
| 2 | L | div | integers | integer* | integer division |
| 2 | L | % | integers | integer* | remainder |
| 3 | L | - | numbers | number* | subtraction |
| 3 | L | + | numbers or strings | number or string* | addition (numbers) or string concatenation |
| 4 | L | << | integers | integer* | bitwise left shift |
| 4 | L | >> | integers | integer* | bitwise right shift with sign |
| 4 | L | >>> | integers | integer* | bitwise right shift with zero fill |
| 5 | L | <, <= | numbers or strings | boolean* | less than, less than or equal |
| 5 | L | >, >= | numbers or strings | boolean* | greater than, greater or equal |
| 6 | L | == | numbers or strings | boolean* | equal (identical values) |
| 6 | L | != | numbers or strings | boolean* | not equal (different values) |
| 7 | L | & | integers | integer* | bitwise AND |
| 8 | L | ^ | integers | integer* | bitwise XOR |
| 9 | L | \| | integers | integer* | bitwise OR |
| 10 | L | && | booleans | boolean* | logical AND |

| | | | | | |
|---|---|---|---|---|---|
| 11 | L | \|\| | booleans | boolean* | logical OR |
| 12 | R | ? : | boolean, any, any | any* | conditional expression |
| 13 | R | = | variable, any | any | assignment |
| 13 | R | *=, -= | variable, number | number* | assignment with numeric operation |
| 13 | R | /= | variable, number | floating-point* | assignment with numeric operation |
| 13 | R | %=, div= | variable, integer | integer* | assignment with integer operation |

[12] Binding: 0 binds tightest

| **Precedence**[13] | **Associativity** | **Operator** | **Operand types** | **Result type** | **Operation performed** |
|---|---|---|---|---|---|
| 13 | R | += | variable, number or string | number or string* | assignment with addition or concatenation |
| 13 | R | <<=, >>=, >>>=, &=, ^=, \|= | variable, integer | integer* | assignment with bitwise operation |
| 14 | L | , | any | any | multiple evaluation |

* The operator can return an invalid value in case the data type conversions fail (see section 6.8 for more information about conversion rules) or one of the operands is invalid.

## 6.2 Functions

A WMLScript function is a named part of the WMLScript compilation unit that can be called to perform a specific set of statements and to return a value. The following sections describe how WMLScript functions can be declared and used.

### 6.2.1 Declaration

Function declaration can be used to declare a WMLScript function name (*Identifier*) with the optional parameters (*FormalParameterList*) and a block statement that is executed when the function is called. All functions have the following characteristics:

- Function declarations *cannot* be nested.
- Function names must be *unique* within one compilation unit.
- All parameters to functions are *passed by value*.
- Function calls must pass *exactly* the same number of arguments to the called

function as specified in the function declaration.
- Function parameters behave like *local variables* that have been initialised before the function body (block of statements) is executed.
- A function *always* returns a value. By default it is an empty string (""). However, a *return* statement can be used to specify other return values.

Functions in WMLScript are not data types[14] but a syntactical feature of the language.

**Syntax:**

*FunctionDeclaration* **:**

**extern**$_{opt}$ **function***Identifier* **(***FormalParameterList*$_{opt}$ **)***Block* **;**$_{opt}$

*FormalParameterList* **:**

*Identifier*

*FormalParameterList* **,***Identifier*

[13] Compatibility note: Functions in ECMAScript are actual data types.

**Arguments**: The optional **extern** keyword can be used to specify a function to be externally accessible. Such functions can be called from outside the compilation unit in which they are defined. There must be at least one externally accessible function in a compilation unit. *Identifier* is the name specified for the function. *FormalParameterList* (optional) is a comma-separated list of argument names. *Block* is the body of the function that is executed when the function is called and the parameters have been initialised by the passed arguments.

**Examples:**

```
function currencyConverter(currency, exchangeRate) { return
   currency*exchangeRate;
};

extern function testIt() { var
   UDS = 10;
   var FIM = currencyConverter(USD, 5.3);
};
```

### 6.2.2 Function Calls

The way a function is called depends on where the called (target) function is declared. The following sections describe the three function calls supported by WMLScript: local script

function call, external function call and library function call.

#### 6.4.2.1 Local Script Functions

Local script functions (defined inside the same compilation unit) can be called simply by providing the function name and a comma separated list of arguments (number of arguments must match the number of parameters[15] accepted by the function).

**Syntax:**

*LocalScriptFunctionCall* **:**

*FunctionName Arguments*

*FunctionName* **:**

*Identifier*

*Arguments* **:**

**(** **)**
**(** *ArgumentList* **)**

*ArgumentList* **:**

*AssignmentExpression*

*ArgumentList* **,***AssignmentExpression*

Functions inside the same compilation unit can be called before the function has been declared:

[14] Compatibility note: ECMAScript supports a variable number of arguments in a function call.

```
function test2(param) {
   return
   test1(param+1);
};

function test1(val) {
   return val*val;
};
```

#### 6.4.2.2 External Functions

External function calls must be used when the called function is declared in an external compilation unit. The function call is similar to a local function call but it must be prefixed with the name of the external compilation unit.

**Syntax:**

*ExternalScriptFunctionCall* **:**

*ExternalScriptName* **#***FunctionName Arguments*

*ExternalScriptName* **:**

*Identifier*

Pragma use url (see section 6.7) must be used to specify the external compilation unit. It defines the mapping between the external unit and a name that can be used within function declarations. This name and the hash symbol (#) are used to prefix the standard function call syntax:

```
use url OtherScript "http://www.host.com/script"; function

test3(param) {
    return OtherScript#test2(param+1);
};
```

#### 6.4.2.3 Library Functions

Library function calls must be used when the called function is a WMLScript standard library function [WMLSLibs].

**Syntax:**

*LibraryFunctionCall* **:**

*LibraryName* **.***FunctionName Arguments*

*LibraryName* **:**

*Identifier*

A library function can be called by prefixing the function name with the name of the library (see section
6.6 for more information) and the dot symbol (.):

```
function test4(param) {
    return Float.sqrt(Lang.abs(param)+1);
};
```

### 6.2.3 Default Return Value

The default return value for a function is an empty string (""). Return values of functions can be ignored (ie, function call as a statement):

```
function test5() {
    test4(4);
};
```

## 6.5 Statements

WMLScript statements consist of expressions and keywords used with the appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line.

The following sections define the statements available in WMLScript[16]: empty statement, expression statement, block statement, break, continue, for, if...else, return, var, while.

### 6.5.1 Empty Statement

Empty statement is a statement that can be used where a statement is needed but no operation is required.

**Syntax:**

*EmptyStatement* **:**

**;**

**Examples:**

```
while (!poll(device)) ; // Wait until poll() is true
```

### 6.5.2 Expression Statement

Expression statements are used to assign values to variables, calculate mathematical expressions, make function calls etc.

**Syntax:**

*ExpressionStatement* **:**

*Expression* **;**

*Expression* **:**

*AssignmentExpression*

*Expression* **,***AssignmentExpression*

---

[15] Compatibility note: ECMAScript supports also *for..in* and *with* statements.

**Examples:**

```
str = "Hey " + yourName;
val3 = prevVal + 4;
counter++;
myValue1 = counter, myValue2 = val3; alert("Watch
out!");
retVal = 16*Lang.max(val3,counter);
```

### 6.5.3 Block Statement

A set of statements enclosed in the curly brackets is a block statement. It can be used anywhere a single statement is needed.

**Syntax:**

*Block* **:**

**{***StatementList*$_{opt}$ **}**

*StatementList* **:**

*Statement*

*StatementList Statement*

### Example:

```
{
    var i = 0;
    var x = Lang.abs(b); popUp("Remember!");
}
```

Variable Statement

This statement declares variables with initialisation (optional, variables are initialised to empty string ("") by default). The scope of the declared variable is the rest of the current function (see section 6.2.2 for more information about variable scoping).

**Syntax:**

*VariableStatement* **:**

**var***VariableDeclarationList* **;**

*VariableDeclarationList* **:**

*VariableDeclaration*

*VariableDeclarationList* **,***VariableDeclaration*

*VariableDeclaration* **:**

*Identifier VariableInitializer*$_{opt}$

*VariableInitializer* **:**

**=***ConditionalExpression*

**Arguments**: *Identifier* is the variable name. It can be any legal identifier. *ConditionalExpression* is the initial value of the variable and can be any legal expression. This expression (or the default initialisation to an empty string) is evaluated every time the variable statement is executed.

Variable names must be unique within a single function.

**Examples:**

```
function count(str) {
```

```
    var result = 0;              //  Initialized  once
    while (str != "") {
        var ind = 0;             // Initialized every time
        // modify string
    };
    return result
};

function   example(param)   {
    var a = 0;
    if (param > a) {
        var b = a+1;             // Variables a and b can be  used
    } else {
        var c = a+2;             // Variables a, b and c can be  used
    };
    return a;                    // Variable a, b and c are  accessible
};
```

### 6.5.5 If Statement

This statement is used to specify conditional execution of statements. It consists of a condition and one or two statements and executes the first statement if the specified condition is *true*. If the condition is *false*, the second (optional) statement is executed.

**Syntax:**

*IfStatement* **:**

**if(***Expression* **)***Statement* **else***Statement*

**if(***Expression* **)***Statement*

**Arguments**: *Expression* (condition) can be any WMLScript expression that evaluates (directly or after conversion) to a *boolean* or an *invalid* value. If condition evaluates to true, the first statement is executed. If condition evaluates to false or invalid, the second (optional) else statement is executed. *Statement* can be any WMLScript statement, including another (nested) if statement. elseis always tied to the closest if.

**Example:**

```
if (sunShines)
{ myDay = "Good"; goodDays++;
} else
```

**Return Statement**

This statement can be used inside the function body to specify the function return value. If no return statement is specified or none of the function return statements is executed, the function returns an empty string by default.

### Syntax:

*ReturnStatement* **:**

**return** *Expression*$_{opt}$ **;**

### Example:

```
function square( x ) {
   if (!(Lang.isFloat(x))) return invalid; return x * x;
};
```

## 6.6 Libraries

WMLScript supports the usage of libraries[17]. Libraries are named collections of functions that belong logically together. These functions can be called by using a dot ('.') separtor with the library name and the function name with parameters:

An example of a library function call:

```
function dummy(str) {
    var i = String.elementAt(str,3," ");
};
```

### 6.6.1 Standard Libraries

Standard libraries are specified in more detail in the *WAP-194-WMLScript Standard Libraries Specification* [WMLSLibs].

---

[16] Compatibility note: ECMAScript does not support libraries. It supports a set of predefined objects with attributes. WMLScript uses libraries to support similar functionality.

## 6.7 Pragmas

WMLScript supports the usage of *pragmas* that specify compilation unit level information. Pragmas are specified at the beginning of the compilation unit before any function declaration. All pragmas start with the keyword use and are followed by pragma specific

attributes.

**Syntax:**

*CompilationUnit* **:**

*Pragmas*$_{opt}$ *FunctionDeclarations*

*Pragmas* **:**

*Pragma*
*Pragmas*
*Pragma*

*Pragma* **:**

**use** *PragmaDeclaration* **;**

*PragmaDeclaration* **:**

*ExternalCompilationUnitPr*
*agma*
*AccessControlPragma*
*MetaPragma*

The following sections contain more information about the supported pragmas.

### 6.7.1 External Compilation Units

WMLScript compilation units can be accessed by using a URL. Thus, each WMLScript function can be accessed by specifying the URL of the WMLScript resource and its name. A use url pragma must be used when calling a function in an external compilation unit.

**Syntax:**

*ExternalCompilationUnitPragma* **:**

**url***Identifier StringLiteral*

The use url pragma specifies the location (URL) of the external WMLScript resource and gives it a local *name*. This name can then be used inside the function declarations to make external function calls (see section 6.4.2.2).

```
use url OtherScript "http://www.host.com/app/script"; function
```

```
test(par1, par2) {
    return OtherScript#check(par1-par2);
};
```

The behaviour of the previous example is the following:

- The pragma specifies a URL to a WMLScript compilation unit.
- The function call loads the compilation unit by using the given URL (http://www.host.com/app/script)
- The content of the compilation unit is verified and the specified function (check) is executed

The use url pragma has its own name space for local names. However, the local names must be unique within one compilation unit. The following URLs are supported:

- Uniform Resource Locators [RFC2396] without a hash mark (#) or a fragment identifier. The schemes supported are specified in [WAE].
- Relative URLs [RFC2396] without a hash mark (#) or a fragment identifier: The base URL is the URL that identifies the current compilation unit.

The given URL must be escaped according to the URL escaping rules. No compile time automatic escaping, URL syntax or URL validity checking is performed.

### 6.7.2 Access Control

A WMLScript compilation unit can protect its content by using an *access control* pragma. Access control must be performed before calling external functions. It is an error for a compilation unit to contain more than one access control pragma.

**Syntax:**

*AccessControlPragma* **:**

**access** *AccessControlSpecifier*

*AccessControlSpecifier*
*er* **: domain**
*StringLiteral*
**path**
*StringLiteral*

**domain** *StringLiteral* **path** *StringLiteral*

Every time an external function is invoked an access control check is performed to determine whether the destination compilation unit allows access from the caller.

Access control pragma is used to specify *domain* and *path* attributes against which these access control checks are performed. If a compilation unit has a domain and/or path attribute, the referring compilation unit's URL must match the values of the attributes. Matching is done as follows: the access domain is suffix-matched against the domain name portion of the referring URL and the access path is prefix- matched against the path portion of the referring URL. Domain and path attributes follow the URL capitalisation rules.

Domain suffix matching is done using the entire element of each sub-domain and must match each element exactly (e.g. www.wapforum.org shall match wapforum.org, but shall not match forum.org).

Path prefix matching is done using entire path elements and must match each element exactly (e.g.
/X/Ymatches /X, but does not match /XZ).

The domain attribute defaults to the current compilation unit's domain. The path attribute defaults to the value "/".

To simplify the development of applications that may not know the absolute path to the current compilation unit, the path attribute accepts relative URLs [RFC2396]. The user agent converts the relative path to an absolute path and then performs prefix matching against the path attribute.

Given the following access control attributes for a compilation unit:

```
use access domain "wapforum.org" path "/finance";
```

The following referring URLs would be allowed to call the external functions specified in this compilation unit:

```
http://wapforum.org/finance/money.cgi
https://www.wapforum.org/finance/markets.cgi
http://www.wapforum.org/finance/demos/packages.cgi?x=123&y=456
```

The following referring URLs would not be allowed to call the external functions:

```
http://www.test.net/finance
http://www.wapforum.org/internal/foo.wml
```

By default, access control is disabled (ie, all external functions have public access).

### 6.7.3 Meta-Information

Pragmas can also be used to specify compilation unit specific meta-information. Meta-

information is specified with property names and values. This specification does not define any properties, nor does it define how user agents must interpret meta-data. User agents are not required to act on the meta- data.

**Syntax:**

*MetaPragma* **:**

**meta***MetaSpecifier*

*MetaSpecifier* **:**

*MetaNa*
*me*
*MetaHttp*
*Equiv*
*MetaUser*
*Agent*

*MetaName* **:**

**name** *MetaBody*

*MetaHttpEquiv* **:**

**http equiv** *MetaBody*

*MetaUserAgent* **:**

**user agent** *MetaBody*

*MetaBody* **:**

*MetaPropertyName MetaContent* $MetaScheme_{opt}$

Meta-pragmas have three attributes: *property name*, *content* (the value of the property) and optional *scheme* (specifies a form or structure that may be used to interpret the property value – the values vary depending on the type of meta-data). The attribute values are string literals.

#### 6.7.3.1 Name

Name meta-pragma is used to specify meta-information intended to be used by the

origin servers. The user agent should ignore any meta-data named with this attribute. Network servers should not emit WMLScript content containing meta-name pragmas.

```
use meta name "Created" "18-March-1998";
```

#### 6.7.3.2 HTTP Equiv

HTTP equiv meta-pragma is used to specify meta-information that indicates that the property should be interpreted as an HTTP header (see [RFC2068]). Meta-data named with this attribute should be converted to a WSP or HTTP response header if the compilation unit is compiled before it arrives at the user agent.

```
use meta http equiv "Keywords" "Script,Language";
```

#### 6.7.3.3 User Agent

User agent meta-pragma is used to specify meta-information intended to be used by the user agents. This meta-data must be delivered to the user agent and must not be removed by any network intermediary.

```
use meta user agent "Type" "Test";
```

## 6.8 Automatic Data Type Conversion Rules

In some cases, WMLScript operators require specific data types as their operands. WMLScript supports automatic data type conversions to meet the requirements of these operators. The following sections describe the different conversions in detail.

### 6.8.1 General Conversion Rules

WMLScript is a weakly typed language and the variable declarations do not specify a type. However, internally the language handles the following data types:

- *Boolean*: represents a boolean value true or false.
- *Integer*: represents an integer value
- *Floating-point*: represents a floating-point value
- *String*: represents a sequence of characters
- *Invalid*: represents a type with a single value invalid

A variable at any given time can contain a value of one of these types. WMLScript provides an operator *typeof*, which can be used to determine what is the current type of a variable or any expression (no conversions are performed).

Each WMLScript operator accepts a predefined set of operand types. If the provided

operands are not of the right data type an automatic conversion must take place. The following sections specify the legal automatic conversions between two data types.

### 6.8.2 Conversions to String

Legal conversions from other data types to string are:

- Integer value must be converted to a string of decimal digits that follows the numeric string grammar rules for decimal integer literals. See section 7.4 for more information about the numeric string grammar.
- Floating-point value must be converted to an implementation-dependent string representation that follows the numeric string grammar rules for decimal floating-point literals (see section 7.1.4 for more information about the numeric string grammar). The resulting string representation must be equal to the original value (ie .5 can be represented as "0.5", ".5e0",etc.).
- The boolean value true is converted to string "true" and the value falseis converted to string "false".
- Invalidcan not be converted to a string value.

### 6.8.3 Conversions to Integer

Legal conversions from other data types to integer are:

- A string can be converted into an integer value only if it contains a decimal representation of an integer number (see section 7.4 for the numeric string grammar rules for a decimal integer literal).
- Floating-point value cannot be converted to an integer value.
- The boolean value trueis converted to integer value 1, falseto 0.
- Invalidcan not be converted to an integer value.

### 6.8.4 Conversions to Floating-Point

Legal conversions from other data types to floating-point are:

- A string can be converted into a floating-point value only if it contains a valid representation of a floating-point number (see section 7.4 for the numeric string grammar rules for a decimal floating-point literal).
- An integer value is converted to a corresponding floating-point value.
- The boolean value trueis converted to a floating-point value 1.0, falseto 0.0.
- Invalidcan not be converted to a floating-point value.

The conversions between a string and a floating-point type must be transitive within the ability of the data types to accurately represent the value. A conversion could result in loss of precision.

### 6.8.5 Conversions to Boolean

Legal conversions from other data types to boolean are:

- The empty string ("") is converted to false. All other strings are converted to true.
- An integer value 0 is converted to false. All other integer numbers are converted to true.
- A floating-point value 0.0 is converted to false. All other floating-point numbers are converted to true.
- Invalidcan not be converted to a boolean value.

### 6.8.6 Conversions to Invalid

There are no legal conversion rules for converting any of the other data types to an invalid type. Invalid is either a result of an operation error or a literal value. In most cases, an operator that has an invalid value as an operand evaluates to invalid (see the operators in sections 6.3.8, 6.3.9 and 6.3.10 for the exceptions to this rule).

### 6.8.7 Summary

The following table contains a summary of the legal conversions between data types:

| **Given \ Used as:** | **Boolean** | **Integer** | **Floating - point** | **String** |
|---|---|---|---|---|
| **Boolean true** | - | 1 | 1.0 | "true" |
| **Boolean false** | - | 0 | 0.0 | "false" |
| **Integer 0** | false | - | 0.0 | "0" |
| **Any other integer** | true | - | floating-point value of number | string representation of a decimal integer |

| **Given \ Used as:** | **Boolean** | **Integer** | **Floating - point** | **String** |
|---|---|---|---|---|
| **Floating - point 0.0** | false | Illegal | - | implementation-dependent string representation of a floating-point value, e.g. "0.0" |
| **Any other floating-point** | true | Illegal | - | implementation-dependent string representation of a floating-point value |

| | | | | |
|---|---|---|---|---|
| **Empty string** | false | Illegal | Illegal | - |
| **Non-empty string** | true | integer value of its string representation (if valid – see section 7.4 for numeric string grammar for decimal integer literals) or illegal | floating-point value of its string representation (if valid – see section 7.4 for numeric string grammar for decimal floating-point literals) or illegal | - |
| **invalid** | Illegal | Illegal | Illegal | Illegal |

## 6.9 Operator Data Type Conversion Rules

The previous conversion rules specify when a legal conversion is possible between two data types. WMLScript operators use these rules, the operand data type and values to select the operation to be performed (in case the type is used to specify the operation) and to perform the data type conversions needed for the selected operation. The rules are specified in the following way:

- The additional conversion rules are specified in steps. Each step is performed in the given order until the operation and the data types for its operands are specified and the return value defined.
- If the type of the operand value matches the required type then the value is used as such.
- If the operand value does not match the required type then a conversion from the current data type to the required one is attempted:
  - *Legal conversion*: Conversion can be done only if the general conversion rules (see section 6.9) specify a *legal* conversion from the current operator data type to the required one.
  - *Illegal conversion*: Conversion can not be done if the general conversion rules (see section 6.9) do not specify a *legal* conversion from the current type to the required type.
- If a legal conversion rule is specified for the operand (unary) or for all operands then the conversion is performed, the operation performed on the converted values and the result returned as the value of the operation. If a legal conversion results in an invalid value then the operation returns an invalidvalue.
- If no legal conversion is specified for one or more of the operands then no conversion is performed and the next step in the additional conversion rules is performed. The following

table contains the operator data type conversion rules based on the given operand data types:

| Operand types | Additional conversion rules | Examples |
|---|---|---|
| Boolean(s) | ▪ If the operand is of type boolean or can be converted into a boolean value[18] then perform a boolean operation and return its value, otherwise<br>▪ return invalid | true && 3.4 => boolean<br>1 && 0 => boolean<br>"A" \|\| "" => boolean<br>!42 => boolean<br>!invalid => invalid<br>3 && invalid => invalid |
| Integer(s) | ▪ If the operand is of type integer or can be converted into an integer value[18] then perform an integer operation and return its value, otherwise<br>▪ return invalid | "7" << 2 => integer<br>true << 2 => integer<br>7.2 >> 3 => invalid<br>2.1 div 4 => invalid |
| Floating-point(s) | ▪ If the operand is of type floating-point or can be converted into a floating-point value[18] then perform a floating-point operation and return its value, otherwise<br>▪ return invalid | - |
| String(s) | ▪ If the operand is of type string or can be converted into a string value[18] then perform a string operation and return its value, otherwise<br>▪ return invalid | - |
| Integer or floating-point (unary) | ▪ If the operand is of type integer or can be converted into an integer value then perform an integer operation and return its value, otherwise<br>▪ if the operand is of type floating-point or can be converted into a floating-point value[18] then perform a floating-point operation and return its value, otherwise<br>▪ return invalid | +10 => integer<br>-10.3 => float<br>-"33" => integer<br>+"47.3" => float<br>+true => integer 1<br>-false => integer 0<br>-"ABC" => invalid<br>-"9e9999" => invalid |

Conversion can be done if the general conversion rules (see section **Error! Reference source not found.**) specify a legal conversion from the current type to the required type.

| **Operand types** | **Additional conversion rules** | **Examples** |
|---|---|---|
| Integers or floating-points | ▪ If at least one of the operands is of type floating-point then convert the remaining operand to a floating-point value, perform a floating-point operation and return its value, otherwise<br>▪ if the operands are of type integer or can be converted into integer values[18] then perform an integer operation and return its value, otherwise<br>▪ if the operands can be converted into floating-point values[18] then perform a floating-point operation and return its value, otherwise<br>▪ return invalid | 100/10.3 => float<br>33*44 => integer<br>"10"*3 => integer<br>3.4*"4.3" => float<br>"10"-"2" => integer<br>"2.3"*"3" => float<br>3.2*"A" => invalid<br>.9*"9e999" => invalid<br>invalid*1 => invalid |
| Integers, floating-points or strings | ▪ If at least one of the operands is of type string then convert the remaining operand to a string value, perform a string operation and return its value, otherwise<br>▪ if at least one of the operands is of type floating-point then convert the remaining operand to a floating-point value, perform a floating-point operation and return its value, otherwise<br>▪ if the operands are of type integer or can be converted into integer values[18] then perform an integer operation and return its value, otherwise<br>▪ return invalid | 12+3 => integer<br>32.4+65 => float<br>"12"+5.4 => string<br>43.2<77 => float<br>"Hey"<56 => string<br>2.7+"4.2" => string<br>9.9+true => float<br>3<false => integer<br>"A"+invalid => invalid |

| | | |
|---|---|---|
| Any | ▪ Any type is accepted | a = 37.3<br>b = typeof "s"<br><br>=> float<br>=> string |

## 6.10 Summary of Operators and Conversions

The following sections contain a summary on how the conversion rules are applied to WMLScript operators and what are their possible return value types.

### 6.10.1 Single-Typed Operators

Operators that accept operands of one specific type use the general conversion rules directly. The following list contains all single type WMLScript operators:

| Operato r | Operand types | Result type[19] | Operation performed |
|---|---|---|---|
| ! | boolean | boolean | logical NOT (unary) |
| && | booleans | boolean | logical AND |
| \|\| | booleans | boolean | logical OR |
| ~ | integer | integer | bitwise NOT (unary) |
| << | integers | integer | bitwise left shift |
| >> | integers | integer | bitwise right shift with sign |
| >>> | integers | integer | bitwise right shift with zero fill |
| & | integers | integer | bitwise AND |
| ^ | integers | integer | bitwise XOR |
| \| | integers | integer | bitwise OR |
| % | integers | integer | remainder |
| Div | integers | integer | integer division |
| <<=, >>=, >>>=, &=, ^=, \|= | first operand: variable second operand: integer | integer | assignment with bitwise operation |
| %=, div= | first operand: variable second operand: integer | integer | assignment with numeric operation |

### 6.10.2 Multi-Typed Operators

The following sections contain the operators that accept multi-typed operands:

| Operato | Operand types | Result | Operation performed |
|---|---|---|---|

| r | | type[20] | |
|---|---|---|---|
| ++ | integer or floating-point | integer/floating-point | pre- or post-increment (unary) |
| -- | integer or floating-point | integer/floating-point | pre- or post-decrement (unary) |
| + | integer or floating-point | integer/floating-point | unary plus |
| - | integer or floating-point | integer/floating-point | unary minus (negation) |
| * | integers or floating-points | integer/floating-point | Multiplication |
| / | integers or floating-points | floating-point | Division |
| - | integers or floating-points | integer/floating-point | subtraction |
| + | integers, floating-points or strings | integer/floating-point/string | addition or string concatenation |

[17] All operators may have an invalid result type.

[18] All operators (unless otherwise stated) may have an invalid result type.

| Operator | Operand types | Result type[20] | Operation performed |
|---|---|---|---|
| <, <= | integers, floating-points or strings | boolean | less than, less than or equal |
| >, >= | integers, floating-points or strings | boolean | greater than, greater or equal |
| == | integers, floating-points or strings | boolean | equal (identical values) |
| != | integers, floating-points or strings | boolean | not equal (different values) |
| *=, -= | first operand: variable second operand: integer or floating- point | integer/floating-point | assignment with numeric operation |

| | | | |
|---|---|---|---|
| /= | first operand: variable second operand: integer or floating- point | floating-point | assignment with division |
| += | first operand: variable second operand: integer, floating-point or string | integer/floating -point/string | assignment with addition or concatenation |
| typeof | any | integer[21] | return internal data type (unary) |
| isvalid | any | boolean[21] | check for validity (unary) |
| ? : | first operand: boolean second operand: any third operand: any | any | conditional expression |
| = | first operand: variable second operand: any | any | assignment |
| , | first operand: any second operand: any | any | multiple evaluation |

## UNIT IV

- XML: Introduction XML: An Eagle's Eye view of XML
- XML Definition
- Life of an XML Document
- Related Technologies
- An introduction to XML Applications
- XML Applications
- XML for XML
- First XML Documents Structuring Data: Examining the Data XMLizing the data
- The advantages of the XML format
- Preparing a style sheet for Document Display.

# XML: Introduction XML: An Eagle's Eye view of XML

## What IsXML?

XML stands for Extensible Markup Language (often miscapi- talized as *eXtensibleMarkup Language* to justify the acronym). XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines a syntax in which other field-specific markup languages can be written.

## XML is a meta-markup language

The first thing you need to understand about XML is that it isn't just another markup language like Hypertext Markup Language (HTML) or TeX. These languages define a fixed set of tags that describe a fixed number of elements.

## XML describesstructure and semantics, not formatting

- The second thing to understand about XML is that XML markup describes a docu- ment's structure and meaning. It does not describe the formatting of

the elements on the page. Formatting can be added to a document with a style sheet. The docu- ment itself only contains tags that say what is in the document, not what the docu- ment looks like.

- By contrast, HTML encompasses formatting, structural, and semantic markup. <B> is a formatting tag that makes its content bold. <STRONG>is a semantic tag that means its contents are especially important. <TD>is a structural tag that indicates that the contents are a cell in a table. In fact, some tags can have all three kinds of meaning. An <H1> tag can simultaneously mean 20-point Helvetica bold, a level 1 heading, and the title of the page.

In XML the same data might be marked up like this:

```
<SONG>
  <TITLE>Hot Cop</TITLE>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

## The Life of an XML Document

XML is, at its root, a document format. It is a series of rules about what XML documents look like. There are two levels of conformity to the XML standard. The first is *well-formedness* and the second is *validity*. Part I of this book shows you how to write well-formed documents. Part II shows you how to write valid documents.

HTML is a document format that is designed for use on the Internet and inside Web browsers. XML can certainly be used for that, as this book demonstrates. However, XML is far more broadly applicable. It can be used as a storage format for word processors, as a data interchange format for different programs, as a means of enforcing conformity with Intranet templates, and as a way to preserve data in a human-readable fashion.

### Editors

XML documents are most commonly created with an editor. This may be a basic text editor such as Notepad or vi that doesn't really understand XML at all. On the other hand, it may be a completely WYSIWYG (What You See Is What You Get) edi- tor such as Adobe FrameMaker that insulates you almost completely from the details of the underlying XML format. Or it may be a structured editor such as Visual XML (http://www.pierlou.com/visxml/) that displays XML documents as trees. For the most part, the fancy editors aren't very useful yet, so this book concentrates on writing raw XML by hand in a text editor.

Other programs can also create XML documents. For example, later in this book, you'll see several XML documents whose data came straight out of a FileMaker database. In these cases, the data was first entered into the FileMaker database. Next, a FileMaker calculation field converted that data to XML. Finally, an AppleScript program extracted the data from the database and wrote it as an XML file. Similar processes can extract XML from MySQL, Oracle, and other databases by using Perl, Java, PHP, or any convenient language. In general, XML works extremely well with databases.

In any case, the editor or other program creates an XML document. More often than not, this document is an actual file on some computer's hard disk, but it doesn't

absolutely have to be. For example, the document may be a record or a field in a database, or it may be a stream of bytes received from the network.

## Parsers andprocessors

An XML parser (also known as an XML processor) reads the document and verifies that the XML it contains is well formed. It may also check that the document is valid, although this test is not required. The exact details of these tests are covered in Part II. If the document passes the tests, then the processor converts the docu- ment into a tree of elements.

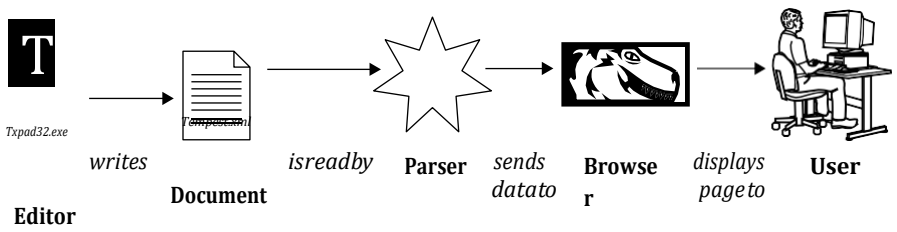
## Browsers and other applications

Finally, the parser passes the tree or individual nodes of the tree to the end application. If this application is a Web browser such as Mozilla, then the browser formats the data and shows it to the user. But other programs may also receive the data.

For instance, a database might interpret an XML document as input data for new records; a MIDI program might see the document as a sequence of musical notes to play; a spreadsheet program might view the XML as a list of numbers and formulas. XML is extremely flexible and can be used for many different purposes.

## The processsummarized

To summarize, an XML document is created in an editor. The XML parser reads the document and converts it into a tree of elements. The parser passes the tree to the browser or other application that displays it. Figure 1-1 shows thisprocess.



## Related Technologies

XML doesn't operate in a vacuum. Using XML as more than a data format involves several related technologies and standards. These include:

- HTML for backward compatibility with legacy browsers
- The CSS and XSL style sheet languages to define the appearance of XML documents
- URLs (Uniform Resource Locaters) and URIs (Uniform Resource Identifiers) to specify the locations of XML documents
- XLinks to connect XML documents to each other
- The Unicode character set to encode the text of XML documents

### HTML

Opera 4.0 and later, Internet Explorer 5.0 and later, Netscape 6.0 and Mozilla pro- vide some (albeit incomplete) support for XML. However, it takes about two years from initial release before most users have upgraded to a particular browser ver- sion (in 2001, my wife still uses Netscape 1.1 on her Mac at work), so you're going to need to convert your XML content into classic HTML for some time to come.

## Cascading Style Sheets

CSS, initially invented for HTML, defines formatting properties such as font size, font family, font weight, paragraph indentation, paragraph alignment, and other styles that can be applied to particular elements. For example, CSS allows HTML documents to specify that all H1 elements should be formatted in 32-point, cen- tered, Helvetica bold. Individual styles can be applied to most HTML elements that override the browser's defaults. Multiple style sheets can be applied to a single doc- ument, and multiple styles can be applied to a single element. The applied styles then cascade according to a particular set of rules.

## Extensible StylesheetLanguage

The Extensible (or eXtensible) Stylesheet Language (XSL) is a more powerful

style language designed specifically for XML documents. XSL style sheets are themselves well-formed XML documents. XSL is actually two different XML applications:

- XSL Transformations (XSLT)
- XSL Formatting Objects (XSL-FO)

Generally, an XSLT style sheet describes a transformation from an input XML docu- ment in one format to an output XML document in another format. That output for- mat can be XSL-FO, but it can also be any other text format, XML or otherwise, such as HTML, plain text, or TeX.

XSL-FO is an XML application that describes the layout of a page. It specifies where particular text is placed on the page in relation to other items on the page. It also assigns styles such as italic or fonts such as Arial to individual items on the page. You can think of XSL-FO as a page description language such as PostScript (minus PostScript's built-in, Turing-complete programming language.)

**URLs andURIs**

XML documents can live on the Web, just like HTML and other documents. When they do, they are referred to by Uniform Resource Locators (URLs), just like HTML files. For example, at the URL http://www.hypermedic.com/style/xml/ tempest.xml you'll find the complete text of Shakespeare's *Tempest* marked up in XML.

Although URLs are well understood and well supported, the XML specification uses Uniform Resource Identifiers (URIs) instead. URIs are a superset of URLs. A *URI* is a more general means of locating a resource; URIs focus a little more on the resource and a little less on the location. Furthermore, they aren't necessarily limited to resources on the Internet. For instance, the URI for this book is uri:isbn:0764547607. This doesn't refer to the specific copy you're holding in your hands. It refers to the almost-Platonic form of the second edition of the *XML Bible* shared by all individual copies.

**XLinks andXPointers**

As long as XML documents are posted on the Internet, people will want to link them to each other. Standard HTML link tags can be used in XML documents, and HTML documents can link to XML documents. For example, this HTML link points to the aforementioned copy of the *Tempest* in XML:

```
<A
  HREF="http://www.hypermedic.com/style/x
  ml/tempest.xml"> The Tempest by
  Shakespeare

</A>
```

**XML Applications**

XML is a meta-markup language for designing domain-specific markup languages. Each specific XML-based markup language is called an *XML application*. This is not an application that uses XML, such as the Mozilla Web browser, the Gnumeric spreadsheet, or the XML Spy editor; rather, it is an application of XML to a specific field such as the Chemical Markup Language (CML) for chemistry or GedML for genealogy.

**Chemical MarkupLanguage**

Peter Murray-Rust's Chemical Markup Language (CML) may have been the first XML application. CML was originally developed as an SGML (Standard Generalized Markup Language) application, and gradually transitioned to XML as the XML stan- dard developed. In its most simplistic form, CML is "HTML plus molecules," but it has applications far beyond the limited confines of the Web.

Mathematical MarkupLanguage

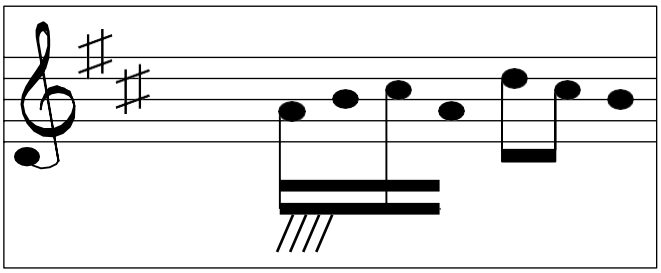
Legend claims that Tim Berners-Lee invented the World Wide Web and HTML at CERN so that high-energy physicists could exchange papers and preprints.Personally, I've never believed that story. I grew up in physics, and while I've wan- dered back and forth between physics, applied math, astronomy, and computer sci- ence over the years, one thing the papers in all of these disciplines had in common was lots and lots of equations. Until now, 10 years after the Web was invented, there hasn't been any good way to include equations in Web pages.

**Vector MarkupLanguage**

Microsoft has developed its own XML application for vector graphics called the Vector Markup Language (VML). VML is supported by Internet Explorer 5.0/5.5 and Microsoft Office 2000. Listing 2-8 is an example of an HTML file with embedded VML that draws the pink triangle. Figure 2-4 shows this file displayed in Internet Explorer 5.5. However, VML is not nearly as ambitious a format as SVG, and leaves out many of the advanced features that SVG includes, such as clipping, masking, and compositing.

**MusicML**

The Connection Factory has created an XML application for sheet music called MusicML. MusicML includes notes, beats, clefs, staffs, rows, rhythms, rests, beams, rows, chords, and more. Listing 2-9 shows the first bar from Beth Anderson's *Flute Swale* in MusicML.



**VoiceXML**

VoiceXML (http://www.voicexml.org/) is an XML application for the spoken word. In particular, it's intended for those annoying voice mail and automated phone response systems. ("If you found a boll weevil in Natural Goodness biscuit dough, please press one. If you found a cockroach in Natural Goodness biscuit dough, please press two. If you found an ant in Natural Goodness biscuit dough, please press 3. Otherwise, please stay on the line for the next available entomologist.")

VoiceXML enables the same data that's used on a Web site to be served up via tele- phone. It's particularly useful for information that's created by combining small nuggets of data,

such as stock prices, sports scores, weather reports, airline sched- ules, and test results. From within the U.S., you can try out some VoiceXML-enabled services by calling 1-800-4-BVOCAL, 1-800-44-ANITA, or 1-800-555-TELL.

## XML forXML

XML is an extremely general-purpose format for text data. Some of the applications it's used for are further refinements of XML itself. These include the XSL style sheet language, the XLink hypertext language, and the XML Schema data descrip- tion language

### First XML Documents

Since this chapter teaches you by example, it will not cross all the *t*s and dot all the *i*s. Experienced readers may notice a few exceptions and special cases that aren't discussed here. Don't worry about these; the details will be covered over the course of the next several chapters. For the most part, you don't need to worry about the technical rules up front. As with HTML, you can learn and do a lot by copying a few simple examples that others have prepared and by modifying them to fit your needs.

### Hello XML

This section follows an old programmer's tradition of introducing a new language with a program that prints "Hello World" on the console. XML is a markup language, not a pro- gramming language; but the basic principle still applies. It's easiest to get started if you begin with a complete, working example that you can build on, rather than starting with more fundamental pieces that by themselves don't do anything.

### Creating a simple XML document

In this section, you create a simple XML document and save it in a file. Listing 3-1 is about the simplest XML document I can imagine, so start with it. This document can be typed in any convenient text editor, such as Notepad, BBEdit, or emacs.

```
<?xml version="1.0"?>
```

```
<FOO>

Hello XML!

</FOO>
```

## Saving the XMLfile

After you've typed in Listing 3-1, save it in a file called hello.xml, HelloWorld.xml, MyFirstDocument.xml, or some other name. The three-letter extension .xml is fairly standard. However, do make sure that you save it in plain-text format, and not in the native format of a word processor such as WordPerfect or Microsoft Word.

# Structuring Data

A document such as this has several potential uses. Most obviously, it can be displayed on a Web page. It can also be used as input to other programs that want to analyze particular seasons or lineups. As the example is developed, you'll learn, among other things, how to mark up data in XML, the principles for good XML element names, and how to prepare a CSS for a document.

### Examining theData

1998 was an astonishing year for baseball. The New York Yankees won their twenty-fourth World Series by sweeping the San Diego Padres in four games. The Yankees finished the regular season with an American League record 114 wins. The St. Louis Cardinals' Mark McGwire and the Chicago Cubs' Sammy Sosa dueled through September for the record, previously held by Roger Maris, for most home runs hit in a single season since baseball was integrated. (The all-time major league record for home runs in a single season is still held by catcher Josh Gibson who hit 75 home runs in the Negro league in 1931. Admittedly, Gibson didn't have to face the sort of pitching Sosa and McGwire faced in today's integrated league. Then again neither did Babe Ruth who was widely — and incorrectly — believed to have held the record until Roger Maris hit 61 in 1961.)

## XMLizing theData

### Starting the document: XML declaration and rootelement

Before I go any further, I'd like to say a few words about naming conventions. As you'll see in Chapter 6, XML element names are quite flexible and can contain any number of letters and digits in either upper- or lowercase. You have the option of writing XML tags that look like any of the following:

<SEASON>

<Season>

<season>

<season1998>

<Season98>

<season_98>

There are several thousand more variations. I don't really care (nor does XML) whether you use all uppercase, all lowercase, mixed-case with internal capitalization, or some other con- vention. However, I do recommend that you choose one convention and stick to it.

## The Advantages of the XML Format

Tables 4-1 and 4-2 do a pretty good job of displaying the batting and pitching data for a team in a comprehensible and compact fashion. What exactly have we gained by rewriting those tables as the much longer XML document of Listing 4-1? There are several benefits. Among them:

- The data is self-describing.
- The data can be manipulated with standard tools.
- The data can be viewed with standard tools.
- Different views of the same data are easy to create with style sheets.

The first major benefit of the XML format is that the data is self-describing. The meaning of

each number is clearly and unmistakably associated with the number itself. When reading the document, you know that the 183 in <HITS>183</HITS> refers to hits and not runs batted in or strikeouts. If the person typing in the docu- ment inadvertently leaves out a statistic, that doesn't mean that every number after it is misinterpreted. HITS is still HITS even if the preceding RUNS element is miss- ing. Another common error in less-verbose formats is transposing values; for instance, using runs for hits and hits for runs. XML lets you transpose with aban- don. As long as the markup is transposed along with the content, no information is lost or misunderstood.

## PreparingaStyleSheetforDocumentDisplay

The view of the raw XML document shown in Figure 4-1 is not bad for some uses. For instance, it allows you to collapse and expand individual elements so you see only those parts of the document you want to see. However, most of the time you'd prob- ably like a more finished look, especially if you're going to display it on the Web. To provide a more polished look, you must write a style sheet for the document.

## Linking to a style sheet

The style sheet can be named anything you like. If it's only going to apply to one document, then it's customary to give it the same name as the document but with the three-letter extension .css instead of .xml. For instance, the style sheet for the XML document 1998shortstats.xml might be called 1998shortstats.css. On the other hand, if the same style sheet will be applied to many documents, then it should probably have a more generic name such as baseballstats.css.

## Assigning style rules to the root element

You do not have to assign a style rule to each element in the list. Many elements can rely on the styles of their parents cascading down. The most important style, therefore, is the one for the root element — SEASONin this example. This defines the default for all the other elements on the page. Computer monitors at roughly 72 dots per inch (dpi) don't have as high a resolution as paper at 300 or more dpi.

Therefore, Web pages should generally use a larger point size than is customary.

# UNIT V

- Attributes, Empty Tags and XSL: Attributes
- Attributes Versus Elements
- Empty Tags
- XSL
- Well formed XML documents
- Foreign Languages and Non Roman Text
- Non Roman Scripts on the Web Scripts, Character sets, Fonts and Glyphs
- Legacy character sets
- The Unicode Character set
- Procedure to Write XML Unicode.

## Attributes, Empty Tags and XSL: Attributes

There are an infinite number of ways to encode any given set of data in XML. . There's no one right way to do it, although some ways are more right than others, and some are more appropriate for particular uses. This chapter explores a different solution to the problem of marking up baseball statistics in XML, carrying over the baseball example from the previous chapter. Specifically, you'll learn how to use attributes to store information and empty-element tags to define element positions. In addition, because CSS doesn't work well with content-less XML elements of this form, we examine an alternative and more powerful style sheet lan- guage called the Extensible Stylesheet Language (XSL).

### Attributes

In the last chapter, all information was provided either in the form of a tag name or as the text content of an element. This is a straightforward and easy-to-understand approach, but it's not the only one possible. As in HTML, XML elements may have attributes. An attribute is a name-value pair associated with an element. The name and the value are each strings.

You're already familiar with attribute syntax from HTML. For example, Cconsider this <IMG>tag:

<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT="Cup

of coffee">

Another difference between HTML and XML is that XML assigns no particular meaning to the IMGelement and its attributes. In particular, there's no guarantee that an XML browser will interpret this element as an instruction to load and dis- play the image in the file cup.gif

### Attributes versusElements

To differentiate between data and metadata, ask yourself whether someone reading the document would want to see a particular piece of information. If the answer is yes, then the information probably belongs in a child element. If the answer is no, then the information probably belongs in an attribute. If all tags were stripped from the document along with all

the attributes, the basic information should still be pres- ent. Attributes are good places to put ID numbers, URLs, references, and other infor- mation not directly or immediately relevant to the reader. However, there are many exceptions to the basic principle of storing metadata as attributes. Reasons for mak- ing an exception include:

- Attributes can't hold structure well.
- Elements allow you to include meta-metadata (information about the informa- tion about the information).
- Not everyone always agrees on what is and isn't metadata.

Elements are more extensible in the face of future changes

**Structured metadata**

Elements can have substructure, attributes can't. This makes elements far more flexible, and may convince you to encode metadata as child elements. For example, suppose you're writing an article and you want to include a source for a fact. It might look something like this:

<FACT SOURCE="The Biographical History of Baseball,

Donald Dewey and Nicholas Acocella (New York: Carroll& Graf Publishers, Inc. 1995) p. 169">

Josh Gibson is the only person in the history of baseballto hit a pitch out of Yankee Stadium.

</FACT>

**Meta-metadata**

Using elements for metadata also easily allows for meta-metadata, or information about the information about the information. For example, the author of a poem may be considered to be metadata about the poem. The language in which that author's name is written is data about the metadata about the poem. This isn't a trivial concern, especially for distinctly non-

Roman languages. For instance, is the author of the *Odyssey* Homer or Όμηρος? Using elements, it's easy write.

```
<POET LANGUAGE="English">Homer</POET>
<POET LANGUAGE="Greek">  Όμηρος  </POET>
```

## What's your metadata is someone else's data

"Metaness" is in the mind of the beholder. Who's reading your document and why they're reading it determines what they consider to be data and what they consider to be metadata. For example, if you're simply reading an article in a scholarly jour- nal, then the name of the author of the article is tangential to the information it con- tains.

## Elements are more extensible

Attributes are certainly convenient when you only need to convey one or two words of unstructured information. In these cases, there may genuinely be no cur- rent need for a child element. However, this doesn't preclude such a need in the future.

## EmptyElementsandEmptyElementTags

An element that contains no content is called an empty element. It can be written like this:

```
<PLAYER          GIVEN_NAME="Rich"
  SURNAME="Becker"
  POSITION="Outfield"      GAMES="79"
  GAMES_STARTED="26"
  AT_BATS="113" RUNS="22" HITS="23"
  DOUBLES="1"              TRIPLES="0"
  HOME_RUNS="3"
  RUNS_BATTED_IN="11"  WALKS="22"
  STRUCK_OUT="34"
  STOLEN_BASES="2"
```

```
CAUGHT_STEALING="0"
SACRIFICE_FLIES="0"

SACRIFICE_HITS="2" HIT_BY_PITCH="2" STEALS="2"></PLAYER>
```

The end tag immediately follows the start tag. Rather than including both a start and an end tag you can include one empty-element tag. Empty-element tags are dis- tinguished from start tags by a closing /> instead of simply a closing >. For instance, instead of <PLAYER></PLAYER> you would write <PLAYER/>.

**XSL**

Attributes are visible in an XML source view of the document as shown in Figure 5-1. However, once a CSS style sheet is applied, the attributes disappear. Figure 5-3 shows Listing 5-1 after the baseball stats style sheet from the previous chapter is applied. It looks like a blank document because CSS styles only apply to element content, not to attributes. If you use CSS, any data that you want to display to the reader should be part of an element's content rather than one of its attributes.

**XSLT templates**

An XSLT style sheet contains templates into which data from the XML document is poured. For example, a template might look like this:

```
<HTML>
  <HEAD>
    <TITLE>
      XSLT Instructions to get the title
    </TITLE>
  </HEAD>
  <BODY>
```

<H1>*XSLT Instructions to get the title*</H1> *XSLT Instructions to get the statistics*

</BODY>

</HTML>

## WELLFORMEDNESS

XML predefines no elements at all. Instead XML allows you to define your own elements as needed. However, these elements and the documents built from them are not completely arbitrary.

Instead, they have to follow a specific set of rules elaborated in this chapter. A document that follows these rules is said to be *well-formed*. Well-formedness is the minimum criteria necessary for XML processors and browsers to read files. This chapter examines the rules for well-formed documents.

It explores the different constructs that make up an XML document — tags, text, attributes, elements, and so on — and discusses the primary rules each of these must follow.

Particular attention is paid to how XML differs from HTML. Along the way I introduce several new XML constructs, including comments, processing instructions, entity refer- ences, and CDATA sections.

This chapter isn't an exhaustive discussion of well-formedness rules. Some of the rules I pre- sent here must be adjusted slightly for documents that have a document type definition (DTD), and there are additional rules for well-formedness that define the relationship between the document and its DTD.

### Well-Formedness Rules

Although XML allows you to invent as many different elements and attributes as you need, these elements and attributes, as well as their contents and the docu- ments that contain them, must all follow certain rules in order to be *well-formed*. If a document is not well-

formed, any attempts to read it or render it will fail.

**XML Documents**

An XML document is made up of text that's divided between markup and character data. It is a sequence of characters with a fixed length that adheres to certain constraints. It may or may not be a file. For instance, an XML document may be:

- A CLOB field in an Oracle database
- The result of a query against a database that combines several records from different tables
- A data structure created in memory by a Java program
- A data stream created on the fly by a CGI program written in Perl
- Some combination of several different files, each of which is embedded in another
- One part of a larger file containing several XML documents

**The XML declaration**

In this and the next several chapters, I treat only simple XML documents that are made up of a single entity, the document itself. Furthermore, these documents only contain text data, not binary data such as images or applets. Such documents can be understood completely on their own without reading any other files. In other words, they stand alone. Such a document normally contains a standalone pseudo-attribute in its XML declaration with the value yes, similar to this one.

## Foreign Languages and Non-Roman Text

Unicode isn't limited to character data either. Non-English characters can be used for markup as well. For example, this is also a well-formed XML document:

料 料 <r□>42</r□>

It's easy to read (or at least look at) these documents in this printed book, but if you were to try displaying them on your computer from the source files, you'd discover they're not quite so simple. You'd very likely end up looking at a screen full of gibberish. Typing them into a text editor would be even more challenging. In this chapter, you learn how international text is represented in computer applications, how XML understands text, and how you can take advantage of the software you already own to read and write in languages other than English.

## Non-Roman Scripts on the Web

Although the Web is international, much of its text is in English. Because of the Web's expansiveness, however, you can still surf through Web pages in French, Spanish, Chinese, Arabic, Hebrew, Russian, Hindi, and other languages. Most of the time, though, these pages come out looking less than ideal. Figure 7-1 shows the October 1998 cover page of one of the United States Information Agency's propa- ganda journals, *Issues in Democracy* (http://usinfo.state.gov/journals/ itdhr/1098/ijdr/ijdr1098.htm), in Russian translation viewed in an English encoding. The red Cyrillic text in the upper left is a bitmapped image file so it's legi- ble (if you speak Russian) and so are a few words in English, such as Adobe Acrobat.

## Scripts, Character Sets, Fonts, and Glyphs

Most modern human languages have written forms. The set of characters used to write a language is called a *script*. A script may be a phonetic alphabet, but it doesn't have to be. For instance, Chinese is written with ideographic characters that represent whole words. Different languages often share scripts, sometimes with slight variations. For instance, the modern Turkish alphabet is essentially the familiar Roman alphabet with three extra letters — g, s, and ı. Chinese and Japanese, on the other hand, share essentially the same 50,000 Han ideographs, although many characters have different meanings and almost all of them have different pronunciations in the different languages. Unfortunately, XML alone is not enough to read a script. For each script a computer processes, four things are required.

1. A character set for the script
2. A font for the character set

**3.** An input method for the character set

An operating system and application software that understand the character set

## A character set for the script

Computers only understand numbers. Before they can work with text, that text has to be encoded as numbers in a specified *character set*. For example, the popular ASCII character set encodes the capital letter A as 65; capital letter B as 66; capital letter C as 67, and so on.

These are semantic encodings that provide no style or font information. **C**, C, or even C are all represented by the number 67. Information about how the character is drawn is stored elsewhere.

## A font for the character set

A font is a collection of glyphs for a character set, generally in a specific size, face, and style. For example, **C**, C, and *C* are all the same character, but they are drawn with different glyphs. Nonetheless, their essential meaning is the same.

Exactly how the glyphs are stored varies from system to system. They may be bitmaps or vector drawings; they may even be hot lead on a printing press. The form they take doesn't concern us here. The key idea is that a font tells the com- puter how to draw each character in the character set.

## An input method for the character set

An input method enables you to enter text. English speakers don't think much about input methods. We just type on our keyboards, and everything's hunky-dory. The same is true in most of Europe, where all that's needed is a slightly modified keyboard with a few extra umlauts, cedillas, or thorns (depending on the country). Figure 7-6 shows a French keyboard I bought in Quebec. It's close to the standard U.S. QWERTY layout.

However, a number of the keys on the right and left have been changed to add French letters and diacritical marks like É and ¨. In addition several keys serve triple duty. The third symbol printed on the lower right hand corner of the key is accessed by holding down the

AltGr (for “Alternate Graphic”) key while pressing the key with the desired character.

**Legacy CharacterSets**

Different computers in different locales use different default character sets. Most modern computers use a superset of the ASCII character set. ASCII encodes the English alphabet and the most common punctuation and white-space characters.

In the United States, Macs use the MacRoman character set, Windows PCs use a character set called Cp1252, and most UNIX workstations use ISO 8859-1, a.k.a Latin-1. These are all extensions of ASCII that support additional characters such as ç and ¿ that are needed for Western European languages such as French and Spanish. In other locales, such as Japan, Greece, and Israel, computers use a still more confusing hodgepodge of character sets that mostly support ASCII plus the local language.

**The ASCII characterset**

ASCII, the American Standard Code for Information Interchange, is one of the original character sets, and is by far the most common. It forms a sort of lowest common denominator for what a character set must support. It defines all the characters needed to write U.S. English, and essentially nothing else. The characters are encoded as the numbers 0 to 127. Table 7-1 presents the ASCII character set.

| *Code* | *Character* | *Code* | *Character* | *Code* | *Character* | *Code* | *Character* |
|---|---|---|---|---|---|---|---|
| 0 | null (Ctrl+@) | 32 | Space | 64 | @ | 96 | ` |
| 1 | start of heading (Ctrl+A) | 33 | ! | 65 | A | 97 | a |
| 2 | start of text | 34 | “ | 66 | B | 98 | b |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | (Ctrl+B) | | | | | | | |
| 3 | end of text (Ctrl+C) | 35 | # | 67 | C | 99 | c | |
| 4 | end of transmission (Ctrl+D) | 36 | $ | 68 | D | 100 | d | |

**The Unicode Character Set**

Using different character sets for different scripts and languages works well enough as long as:

1. You don't need to work in more than one script at once.
2. You never trade files with anyone using a different character set.

Because Macs and PCs use different character sets, more people fail these criteria than not. Obviously what is needed is a single character set that everyone agrees on and that encodes all characters in all the world's scripts. Creating such a set is difficult. It requires a detailed understanding of hundreds of languages and their scripts. Getting software developers to agree to use that set once it's been created is even harder. Nonetheless work is ongoing to create exactly such a set called Unicode, and the major vendors (Microsoft, Apple, IBM, Sun, Be, and many others) are slowly moving toward complying with it. XML specifies Unicode as its default character set.

**How to Write XML in Unicode**

Unicode is the native character set of XML, and XML browsers do a pretty good job of displaying it, at least within the limits of the available fonts. Nonetheless, there simply aren't many, if any, text editors that support the full range of Unicode.

Consequently, you'll probably have to tackle this problem in one of these ways

1. Write in a localized character set such as Latin-3, and then convert your file to Unicode.
2. Include Unicode character references in the text that numerically identify particular characters.

The first option is preferable when you've got a large amount of text to enter in essentially one script, or one script plus ASCII. The second works best when you need to mix small portions of multiple scripts into your document.

**Converting to and from Unicode**

Application software that exports XML files, such as Adobe FrameMaker, handles the conversion to Unicode or UTF-8 automatically. Otherwise, you need to use a conversion tool. Sun's freely available Java Development Kit (http://java. sun.com/j2se/) includes a simple command-line utility called native2ascii that converts between many common and uncommon localized character sets and Unicode.

For example, the following command converts a text file named myfile.txt from the platform's default encoding to Unicode:

```
C:\> native2ascii myfile.txt myfile.uni
```

You can specify other encodings with the -encodingoption.

```
C:\> native2ascii -encoding Big5 chinese.txt chinese.uni
```

You can also reverse the process to go from Unicode to a local encoding with the-reverse option.

```
C:\> native2ascii -encoding Big5 -reverse
chinese.uni chinese.txt
```