WAP Overview

Ric Howell, Chief Technology Officer, Concise Group Ltd.

WAP (the **Wireless Application Protocol**) is a protocol for accessing information and services from wireless devices. WAP is defined and coordinated by the **WAP Forum**, a consortium of industry players who have an interest in extending the kind of information and services that we have become used to accessing over the Internet, to users of mobile devices, including mobile phones. Founded by Phone.com, Ericsson, Nokia and Motorola, the members of the WAP Forum now include most of the leading corporations in the industry, including all the major handset manufacturers, network operators, and software companies.

The objective is to define a standard application framework that will be universal, and that will allow seamless interoperability of all of the components required for mobile access to network applications. The membership of the WAP Forum is both large enough and significant enough to allow them to achieve this goal.

From the start, the WAP Forum has aimed to define a standard that leverages existing Internet technologies wherever possible, and that interoperates with Internet technologies.

Why Do We Need Another Protocol?

Most people would agree that there are more protocols associated with computing than you could shake a stick at, and it can be challenging trying to keep up with them all. So why do we need another protocol? The answer is very simple – because phones are not PCs.

To be specific: most of the protocols in use today make a set of assumptions about the environment, such as the type of network that will be available (particularly from the point of view of bandwidth and reliability), the types of devices that will be accessing the services, and the types of services that will be accessed. These assumptions do not necessarily hold true in the wireless world.

There are a number of differences in terms of the device itself:

> ➢ **Form Factor** — A mobile device needs to be small enough to move around, and ideally to be able to fit in the palm of your hand or carry in a shirt pocket.
> ➢ **CPU** — In a mobile device, the CPU is not nearly as powerful as a desktop PC, and is almost certainly of a different architecture.
> ➢ **Memory and storage** — This is a lot more constrained than on a PC, because handset manufacturers are cost-sensitive, and thus reluctant to add any additional components unless it is really necessary. Also some mobile devices do not have a persistent storage of their own.
> ➢ **Battery** — Mobile devices are battery powered, and the need to have the device available for long periods of time means that the processing CPU cannot make significant demands on the battery.
> ➢ **Display** — This is typically limited in size and resolution, and often cannot cope with color.
> ➢ **Input** — Mobile devices typically do not have keyboards, or if they do they are limited in size. Therefore, input is more challenging than on a typical PC.

A wireless network is considerably different to a fixed-wire network. The bandwidth of the network is typically much smaller, at least at this point in time. Reliability profiles are considerably different, particularly where users move in and out of coverage areas, disappear into tunnels, and so on. Latency may also be an issue in wireless networks. An additional factor is that there are a number of mobile network standards in place across the world, and they do not interoperate seamlessly. Some countries even have incompatible standards in different regions.

Finally, it is important to realize that the market is different where wireless applications are concerned. The types of applications that are suitable for use on mobile devices are not the same as those that are popular on fixed-wire environments. Typical users of mobile applications are likely to be a broader subset of the population than PC users. Even the context in which the applications are going to be used will be different. This highlights the most important aspect of mobile application design, which is to make the application easy to use in the context, and on the device that it will be accessed from.

How Does WAP Address These Issues?

WAP was designed specifically to address these issues, and has implications in each of the above areas.

On the device, the WAP standard defines a **Wireless Application Environment (WAE)**, which is suited to the constraints of mobile devices. The WAE includes a **microbrowser**, which is a **markup language** browser. This browser is less stringent than existing browsers on PCs in terms of specifying exactly how a **User Interface (UI)** element is to be rendered, and concentrates instead on the functionality that is made available through the element (although the markup language itself has to be well formed). Hints can be provided to the microbrowser, but it is up to the microbrowser to select an appropriate representation for the device. WAP also defines a micro **Virtual Machine (VM)** for the microbrowser's scripting language to execute in, which is suited to the memory and CPU constraints of mobile devices. There is also an optimized protocol stack for accessing the network.

The network issues are addressed largely through the protocol stack that was designed to take into account bandwidth limitations and reliability issues. To maintain compatibility and use existing standards where possible, it operates over IP networks, and uses User Datagram Protocol (UDP) over IP wherever possible. However, because the existing mobile networks are not packet switched, it is capable of operating over non-IP networks as well. To help address the bandwidth issue, the content that is transmitted is encoded and compressed to reduce the overall volume of data.

A markup language, **Wireless Markup Language (WML)**, has been defined that is better adapted to the constraints that we previously discussed of mobile devices and wireless networks than HTML. HTML is fairly strongly oriented towards the visual aspects of document rendering and what the specific user interface elements should be and should look like. While this is fine on devices that are capable of sophisticated rendering and have the capability to both render and allow the user to interact with elements, such as push buttons and framesets, it is not appropriate for most mobile devices, and phones in particular. A smaller, tighter markup language was required that is more appropriate to the wireless environment.

WML has been derived from XML and contains elements that more conveniently map to mobile devices than HTML elements. For example, WML defines an `<option>` element, which the microbrowser can render in any appropriate way that is semantically equivalent to the HTML `<button>` element. There is also a scripting language **WMLScript**, which is derived from the standard ECMAScript. Again,

compatibility has been maintained wherever possible, and much of the semantics uses existing HTTP 1.1.
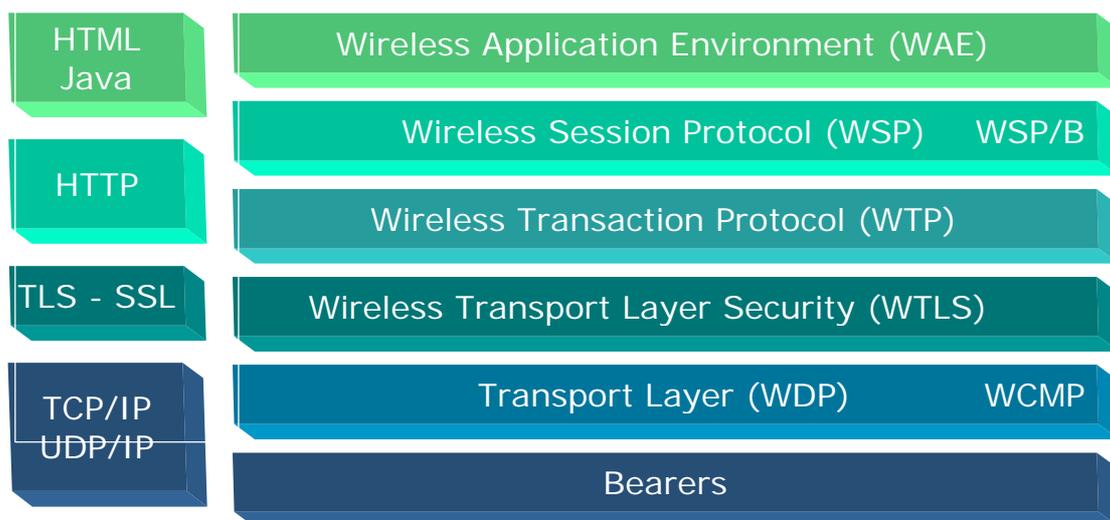
How Does WAP Fit Together?

All of these elements are brought together to provide an environment that facilitates mobile devices interacting with applications that reside on remote servers. The WAE is deployed on the client device to provide an environment in which the user interface can function. Sessions from the client device through to the WAP gateway are established and managed using **Wireless Session Protocol (WSP)/Wireless Transport Protocol (WTP)** and the underlying protocol layers.

The **WAP gateway** itself includes a variety of functionality, including protocol adapters to allow translation between WSP and the various protocols that could be used to access the network services, and encoding and compilation of the WML and WMLScript from text to binary formats. The binary format is used to minimize the amount of data that has to be sent over the network, and also to reduce the amount of processing that the mobile device needs to perform. Although WSP is used almost exclusively by the mobile device to request content from the gateway, the protocol that is used by the back-end service is the one encoded into the URL. The only protocol supported for access to network services at the moment is HTTP.

The WAP gateway communicates with an ordinary web server, using HTTP. What happens from the web server through to the application is no different from Internet access from fixed-wire devices, so the content could be static WML pages, or dynamic content generated using servlets, ASP, CGI, or any other server-side web technology. All content is referenced using standard URLs.

How Does WAP Stack Up?

The differences between the **WAP protocol stack** and a typical Internet protocol stack is the most important part of enabling wireless access for mobile devices. The WAP protocol stack does not map directly onto other stacks, although some comparison is possible. This is illustrated in the diagram below:

HTML Java	Wireless Application Environment (WAE)	
	Wireless Session Protocol (WSP)	WSP/B
HTTP	Wireless Transaction Protocol (WTP)	
TLS - SSL	Wireless Transport Layer Security (WTLS)	
TCP/IP UDP/IP	Transport Layer (WDP)	WCMP
	Bearers	

The kind of functionality that is provided by HTML and Java in the Internet world is incorporated in WAP in the WAE and, to some extent, the WSP. WSP and WTP between them cover the functionality that is provided by HTTP.

If security is required in the fixed-wire world, Transport Layer Security (TLS) is usually used, and in the wireless world there is the **Wireless Transport Layer**

Security (WTLS) equivalent. The transport layer in the fixed-wire world is usually either TCP or UDP over IP, and in the wireless world it is UDP over IP where possible, or the **Wireless Datagram Protocol (WDP)** is provided for networks that cannot support IP at the network layer. Underlying WDP there is a large number of over-the-air bearers that are supported.

Wireless Application Environment

The WAE defines the application environment, which operates on the mobile device, largely in terms of services and content formats.

The architecture of the WAE presumes the existence of **user agents** for interpreting the content referenced by a URL. User agents are required for handling two types of content: the microbrowser for rendering the WML and the virtual machine for execution of the WMLScript. WMLScript also includes a number of standard libraries that provide native implementations of common functions. The specification itself does not mandate a specific implementation or specify any particular user agent.

The WAE provides support for a number of standard content types, including **vCard** and **vCalendar**, for communicating phonebook and calendar information between applications, and **WBMP**, which is a wireless bitmap format.

The Wireless Telephony API provides a mechanism to interact with the device itself and with services and facilities that may be available on the operators network, for example, the ability to access the phone book on a mobile phone.

Wireless Session Protocol

The WSP is responsible for managing the session between the mobile device and the server, and it provides a generalized session management capability. This includes the ability to suspend and resume sessions, which is an important feature of WAP, because connections may be dropped and need reconnecting at a later point in time to continue the session from where it was left off. We have all experienced poor quality reception with mobile phones, connections that drop in mid-sentence, and the classic situation of driving down the road and out of the reception area of the cell. It also provides support for long-lived sessions.

The capability of mobile devices varies significantly, so it follows that the server cannot treat all devices the same way. Part of the session setup process is the negotiation of the nature of the session and the capability of both the participants.

WSP supports both connection-oriented and connectionless sessions, where the connectionless session does not have the overhead of session setup and tear down. Connection-oriented sessions run over WTP, while connectionless sessions use WDP directly.

The encoding and decoding of content is also handled at this level.

Wireless Session Protocol/Browsing

The browsing extensions to WSP add features that are suited to the way that the protocol is likely to be used — browsing WAP sites. The semantics are based on HTTP 1.1. It includes features such as the transmission of session and content headers and the encoding of those headers to make them more efficient for over-the-air-transmission. The content headers (also called Accept headers) define the character sets, language and the user agent profile, which defines the characteristics of the device. The user agent profile information must be passed through to the origin server.

It also provides some **push** capabilities, which are part of the push framework of WAP 1.2. Three levels of push capability are defined: confirmed push within a session context, non-confirmed push within a session, and non-confirmed push without a session. There is also support for asynchronous requests.

Wireless Transaction Protocol

The WTP provides a request/reply based transport mechanism. It includes many of the facilities that are typical of reliable transport protocols, such as retransmission and selective retransmission of lost or corrupted messages, message concatenation, and so on. For the purpose of efficiency, it excludes the explicit session setup and tear down semantics typical of reliable transports. It also provides an abort capability, where a message can be sent to the server to indicate that the results of any processing underway are no longer required.

To match the different requirements of different types of applications, three classes of transport are defined:

> **Class 0** is much like UDP in the fixed-wire world, in that it provides an unreliable, stateless connection, in which no result is communicated and the abort function is not supported.
> **Class 1** defines a reliable stateful transport, which supports retransmission, but does not transmit a result. The abort function is supported.
> **Class 2** is also reliable and stateful, but returns a result for every message sent. This class also supports a 'hold on' reply as a result, which indicates that the server is busy processing the request.

Wireless Transport Layer Security

The wireless equivalent of TLS is WTLS. It provides secure transmission of data across the air network, and supports privacy, message integrity and authentication functions. It is based on its Internet equivalent, but can operate over unreliable transports, in particular WDP and UDP.

Several key exchange suites are included, as well as a number of symmetric and asymmetric encryption ciphers and signature functions. Because much of its authentication is certificate based it can be integrated with **Public Key Infrastructure (PKI)**.

Transport Layer

The preferred transport layer is UDP over IP. However, many of the mobile networks cannot support IP at this point in time, so there is an additional protocol that allows transmission of packets over circuit-switched networks. This is the Wireless Datagram Protocol. It provides a connectionless non-reliable transport protocol with support for message segmentation and reassembly, and UDP-based port numbers. It also provides a consistent interface to all of the underlying over-the-air bearers.

Because WDP is an unreliable transport protocol, it needs a message protocol to signal error conditions, which is the Wireless Control Message Protocol. WCMP is similar to ICMP from the fixed-wire world, and performs similar functions.

Bearers

Underlying the entire protocol stack is a number of over-the-air bearers that can be used to communicate with the mobile device. Some of these, such as GSM, IS-136, CDMA and iDEN, you may be familiar with. All of the most popular standards are

supported, including some that you possibly wouldn't expect, like **Short Message Service (SMS)**.

What is WML?

WML is a generalized markup language that is optimized for limited capability devices and networks. WML documents are an XML document type. It has also borrowed from HDML 2.0 (Phone.com's proprietary markup language) and HTML. It is case sensitive.

WML is based on a **deck** of **cards** metaphor, in which a document is analogous to a deck, and a card is approximately analogous to an individual screen or unit of display. The unit of transmission between the gateway and the mobile device is the deck, and the unit of user interaction is a card within the deck. This structure is illustrated in the diagram below:
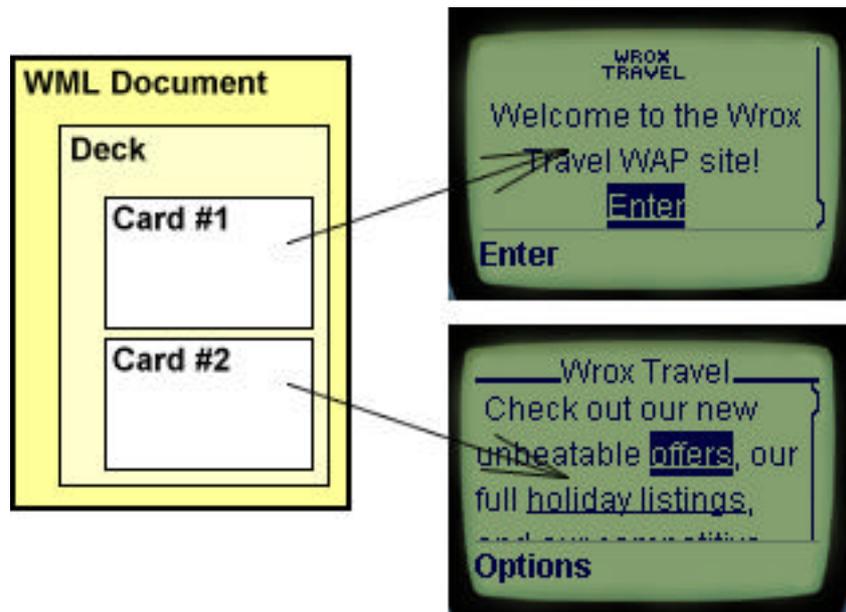
Rather than focusing on the details of the rendering of UI elements, or of how the user should interact with the browser, WML focuses on the semantic meaning of the element. Separating the rendering from the meaning allows the actual rendering and implementation on the device to be adapted to the capabilities of the device.

WML elements support a number of features including text and images, the ability to interact with the user, navigation capabilities and variables. Layout and presentation hints can be included with text and images, but it is ultimately up to the browser how it renders the content. Templates can be used to specify a set of characteristics that apply to all cards in the deck. Examples of WML are given in the next section.

User interaction is facilitated in the form of entering text or selecting options or actions, which trigger events.

Navigation between cards within the deck is supported, as well as between decks. Navigation is either through URL hyperlinks or through a history stack. A URL is taken to reference the first card in a deck, although other cards within the deck can be referenced using fragment anchors. Relative URLs are supported. A history of URLs is maintained in a stack that supports typical stack-like functions (push, pop and reset).

WML supports navigation in both a forward and backward direction through triggering of events. The `onenterforward` event causes the browser to go to the

associated URL if the card is entered in the forward direction, using the `<go>` element or a semantically equivalent element. The URL of the current card is placed on the history stack and can be returned to by using backward navigation. The `onenterbackward` event performs a similar function when the card is entered from the backwards direction, using the `<prev>` element or a semantically equivalent element. The `ontimer` event will cause a jump to the associated URL when a timer expires. The `onpick` event is triggered when an item is selected from a list.

The actions to be performed in response to events and user interactions are specified in terms of tasks. The `<go>` element indicates navigation to a URI and pushes the current URI onto the history stack. An HTTP submission method of `post` or `get` can be specified as well as the variables that are to be transmitted to the server as part of the URI request. Backward navigation through the history stack is specified with the `<prev>` element. With either a `<go>` or `<prev>` a variable value can be set. The `<refresh>` element is used to cause the display to be refreshed with the value of variables that have changed since the last display. The `<noop>` element causes nothing to happen and is used to override deck level elements in a card.

Tasks need to be bound to events so that the tasks execute when the event is triggered. The `<do>` element specifies a user interface element that the user can directly interact with to trigger the event. The rendering of the UI element is up to the browser, although a hint is given as to the semantic meaning of the element through the `type` attribute. The `<do>` element can occur at either the card or deck level, and may be designated as optional.

The `<onevent>` element is specified within an enclosing element and is used to bind a task to an intrinsic event for the enclosing element. For example, an `<onevent>` element enclosed within a `<card>` element, with a type of `onenterforward` and which encloses a `<go>` element, will bind the `<go>` element to the `onenterforward` event for that card. When someone enters that card in the forward direction the `<go>` will be triggered and the URL specified in the `href` attribute of the `<go>` element will be loaded.

The WML browser maintains a context that can contain variables. Variable names are case sensitive and are preceded by a `$`. Parameterized cards and variable substitution allows for the contents of the card to be customized on the phone, but variables cannot be substituted for elements or attributes. Variables can be shared between cards and decks. The value of a variable is set by a `<setvar>` element, an `<input>` element or a `<select>` element. A `<postfield>` element can be used to transmit the contents of a variable to the server during a URL request.

User input is facilitated through the `<input>` and `<select>` elements. The `<input>` element specifies a text entry object. As before, the form and rendering of the object depends on the browser. A default value can be specified, and the length of the object, the maximum size of the text entered and formatting options can also be specified. Empty strings can be refused and password masks can be applied to the input. The `<select>` element indicates a choice from a list of options. Options can be grouped to form hierarchies, and both single and multiple-choice lists are supported. A default can also be specified for use in the event that the user does not make a selection.

Links are specified with either the `<anchor>` element or the `<a>` element, which is a short form of `<anchor>`. The element specifies the head of the link — the tail is always specified by another element. Anchors must have an associated task. The link can specify an image that is to be included in the document. It can include some alternate text to be used if the image cannot be displayed, and it can also have dimensions and alignment specified.

The `<timer>` element is used to set the value of a timer on entry to a card. The time is specified in tenths of a second and the `ontimer` event will be triggered when the timer reaches 0.

Tables are supported through the `<table>`, `<tr>` and `<td>` elements. The number of columns in the table is specified as an attribute of the `<table>` element. The `<tr>` element contains a row of the table, and the `<td>` element contains a cell within the table.

What Does WML Look Like?

This section contains some examples of WML, along with screenshots showing how the card is rendered in a WML browser. Working through this section should help to clarify the use and effect of some of the WML elements mentioned above.

The Document Prologue

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

<wml>
...
</wml>
```

Defining a Template

```
<wml>
    <template>
        <do type="goBack" name="goBack" label="Back">
            <prev/>
        </do>
    </template>
    <card id="init" newcontext="true">
    </card>
</wml>
```
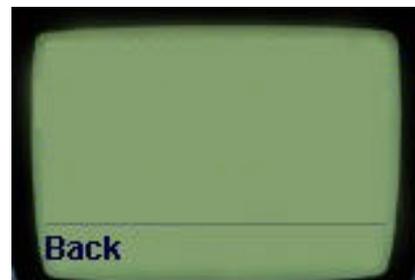
The above snippet of WML results in the Back option being displayed on each card in the deck. It uses the `<do>` element to associate the label `Back` with the `<prev>` task.

Whenever the user selects the `Back` option, the microbrowser will navigate to the previous card:

Defining a Card

```
<wml>
    <card id="init" newcontext="true">
        <p align="center">
            <b>Portfolio Service</b><br/>
        </p>
    </card>
</wml>
```

This code snippet just shows a simple WML card. It includes some formatting elements to influence how the text is laid out by the microbrowser. The card is shown here:

Using Anchors

```
<wml>
<card id="MainMenu" title="Main Menu">
   <p>
      <anchor>1. Market Indices
         <go href="#MarketIndices"/>
      </anchor><br/>
      <anchor>2. Portfolio Valuation
         <go href="#PortfolioValuation"/>
      </anchor><br/>
      <anchor>3. Current Fund Prices
         <go href="#CurrentFundPrices"/>
      </anchor><br/>
      <anchor>4. Buy/Sell
         <go href="#BuySell"/>
      </anchor><br/>
   </p>
</card>
</wml>
```

Anchors are used to link to other cards, as shown in the code snippet above and the screen shot here. Each anchor element is bound to a `<go>` task that references another card in the deck.

Using Input Fields

```
<wml>
<card id="Login" title="Client Login">
   <p>
      Please enter:<br/>
      Account No
      <input type="text" name="txtAccountNo"/><br/>
      Security Code
      <input type="password" name="txtSecurityCode"/><br/>
   </p>
   <do type="goMainMenu" label="Main Menu">
      <go href="#MainMenu"/>
   </do>
</card>
</wml>
```

Input fields are used to allow the user to enter data. The code snippet above creates two input fields, one of which takes some text, the other of which takes a password. The password will be obscured by asterisks as it is rendered on the screen. The card is also bound to a `<go>` task, which links it to the main menu, by a `<do>` element.

Using Tables and Graphics

```
This final example shows the use of graphics and tables.
<wml>
<card id="init" newcontext="true">
   <p align="center">
      <table columns="1">
         <tr><td>
            <img src="abcbank.gif" alt="ABC Bank"/><br/>
         </td></tr>
      </table>
      <b>Portfolio Service</b><br/>
   </p>
   <do type="goLogin" label="Login">
      <go href="#Login"/>
   </do>
</card>
</wml>
```

This final example shows the use of graphics and tables. A table containing one column is created using the `<table>` element. The `<p>` element is used to align the table in the center of the screen. Within the table a `<tr>` element creates a single row that contains a single cell, created by the `<td>` element. Inside the cell the `<img>` element creates a link to an image, with some alternate text associated with it for use in the event that the image cannot be displayed. A `<do>` element binds the card to a `<go>` task that links to the login card within the deck.

What is WMLScript?

WMLScript is to WML what JavaScript is to HTML; it is a scripting language, which (like JavaScript) is based on ECMAScript, although it is not fully compliant with ECMAScript. It includes all of the usual procedural constructs that you would expect in a scripting language, such as loops, conditions, and so on.

For the purposes of efficiency, it is compiled on the server into byte code, which is then executed on the mobile device in the VM. Each compilation unit is referenced by a URL, with externalized functions within the compilation unit referenced by fragment anchors.

Functions are also built into WMLScript, and additional functions are available in the standard WMLScript libraries, which include functions for language, string manipulation, URL handling, an interface to the browser itself, floating points, and dialogs. There is also a cryptography library, but it only contains a function for signing plaintext at this time.

WMLScript is weakly typed, meaning that type checking is not enforced and a variable's type may change during its lifetime. Variables must be declared, but the data type isn't. The types supported are Boolean, integer (32 bit), floating-point (32 bit single precision), string, and invalid. Automatic conversion between data types is supported wherever possible.

The typical operations that you would expect in any scripting language are supported. This includes the assignment operators, arithmetic operators, logical operations, string manipulation and comparison operators. Arrays are not directly supported, but the string functions support indexing. Two operators, `typeof` and `isvalid` are used for type checking.

Functions are declared with a name, the parameters that the function accepts, and a block statement that contains the code. Parameters are passed by value and are treated as local variables. All functions always return a value, and the default value is an empty string. Function declarations cannot be nested. Local functions, that is, functions within the same compilation unit, are referenced directly by name, whereas external functions are called using a URL and fragment anchor. Standard library functions are called using the library name and function name separated by a period. The code snippet in the next section contains an example of calling a standard function in this way.

Conditional branching is limited to `if` and `if else` statements. Both `while` and `for` statements are supported for looping, with `break` and `continue` statements to control program flow in loops.

WMLScript can be used to add a level of intelligence and processing capability to WML pages. Used appropriately, it can help to reduce the amount of network transmission in an application by performing some input validation on the mobile device. It is typically invoked in response to events on the mobile device.

WML variables can be used in WMLScript.

What Does WMLScript Look Like?

The snippet of WMLScript below shows a very simple function that maps a printer's name to a printer code.

```
/*
 * Given a printers name, put the printer's
 * code in the variable returnPrinter.
 *@param printer the variable name in which to store the printer code
 *@param printerName the printer's name
*/
extern function getPrinter(printer, printerName)
{
  var returnPrinter ="Printer Unknown ...";

  if ("First Floor Laser 1" == printerName) {
    returnPrinter = "FFLPRNT01";
  }
  else if ("First Floor Laser 2" == printerName) {
    returnPrinter = "FFLPRNT02";
  }
  else if ("Second Floor Laser 1" == printerName) {
    returnPrinter = "SFLPRNT01";
  }
  else if ("Colour Printer" == printerName) {
    returnPrinter = "CLRPRNT";
  }
  else if ("CEO's Printer" == printerName) {
    returnPrinter = "CEOPRNT";
  }

  WMLBrowser.setVar(printer, returnPrinter);

  /*
    * Make sure the browser updates its display.
  */
  WMLBrowser.refresh();
};
```

The code shows an example of passing parameters into a function and setting a variable. `if else` statements are used to perform a very simple translation. Note the use of the `refresh` function from the `WMLBrowser` standard library to update the display at the end of the function.

What is WTA?

The Wireless Telephony API is an API and framework for accessing telephony and network services. It is intended mostly for use by network operators to control access to resources on the network from a mobile device, and to modify the way the device responds to events on the network.

Event responses are governed by an event table, and the WTA provides a means to interrogate and update this event table. Developers can use it to access a subset of common, non–privileged network services in a network-independent way. An example would be the ability to initiate a telephone call. The facilities are made available though libraries of common services, network specific services and public services. Only the public services are available to content that is not running within the WTA user agent.

Execution of privileged WTA content takes place in a separate user agent that is accessed via its own port under the WTLS protocol.

Summary

In this paper we have taken a brief look at WAP and its constituent components, what issues it tries to address and how it does this, and what facilities are available for the developer to build applications that are appropriate for deployment on mobile devices and across wireless networks. It is clear that the WAP Forum have made intelligent and appropriate architectural choices in designing a protocol stack and architecture that successfully leverages Internet technologies, thereby allowing us to transfer our skills to the new environment without a significant learning curve. At the same time their architectural choices go a long way to addressing the concerns and issues that arise from mobile computing. In choosing to implement a new mark-up language they have somewhat simplified the task of creating applications that are appropriate to the devices on which they will execute. It is an obviously better solution than trying to adapt HTML for this purpose. The scripting capabilities provided by WMLScript provide a powerful medium for extending the ability of applications and adding a layer of intelligent processing on the device. WAP provides a platform that is easily accessible and almost invites you to develop applications for it!