The Wireless Application Protocol (WAP) is emerging as a standard wireless protocol and browser framework for small, limited display devices. WAP is an Internet-enabling protocol, based on XML, that allows data access for cell phones, PDAs, and other low-horsepower systems. WAP is a reaction to two phenomena: the explosive growth of the World Wide Web and digital wireless devices.

Access to the Internet and Web on the desktop is a fairly mature technology, even despite its short existence. For the everyday user, modem speeds are tolerable, access is better than ever, and a large number of Internet service providers make mass access affordable. For companies and power users, technologies such as Digital Subscriber Lines, or DSL, offer ultra high speed, always-connected Internet access. This isn’t the case with wireless Web access. Wireless Web is new, immature, and inherently slow. Desktop browsers, such as Netscape Navigator and Internet Explorer easily support complex text and graphics and render them relatively quickly, but the opposite is true with a wireless device. Wireless devices are slow and display-handicapped, but mobile Web access is still a great convenience. Several wireless phone manufacturers recognized this early and formed the WAP Forum (www.wapforum.org) to create a specification for a Web access environment based on current Internet and Web technology.

Clients, servers, and proxies
Information on the Web is manipulated by various technologies organized into three basic categories: clients, servers, and proxies. Clients are the programs you use. They allow you to traverse the Web and retrieve and submit information. Servers contain information available to clients all over the world. Proxies act as intermediaries between clients and servers.
Typical client applications are web browsers such as Netscape Navigator and Internet Explorer. Browsers make information available to a user by navigating through hypertext links and forms. Unseen client applications may contain Web-enabled connectivity without the user really knowing it. An example is an application that logs data from a remote location and posts the data on a Web page. The client application accesses the data posted on the Web page and presents the data to the user in what appears to be a stand-alone application.

Servers process requests from multiple clients across the Web. Desktop clients typically request servers to retrieve Web pages, navigate hypertext links, and access databases. Database access is achieved via the Common Gateway Interface, or CGI. A server invokes a CGI script using client data to access, process, and return data to the client.

A proxy acts on behalf of the clients it serves, and behaves as a sort of funnel. Clients usually reside behind firewalls. Proxies receive requests from clients behind a firewall and wait for responses outside the firewall. All transactions go through the proxy, and proxies may flag or reject some transactions. Proxies can also be advantageous, especially in intranets. An intranet is an internal network that utilizes Internet protocols. Proxies can cache documents and Web pages, allowing faster access to frequently requested information.

**Web browsing on wireless non-PCs**

Web technology is fairly mature on PCs and workstations, mostly due to the capability and speed of desktop machines and the rapidly evolved features supported by HTML. The Internet and the World Wide Web are based on wired technology.

It’s a different story on non-PCs, such as cellular phones and wireless PDAs. The computational power and display capabilities of handheld wireless devices are feeble compared to desktop devices. Having a four-line LCD display on a cellular phone with primitive graphics capability seems extravagant. PDAs are mild exceptions with bitmapped images and touch-screens. A desktop computer with a Megabit connection makes the Internet seem blazingly fast, but be aware that a cellular phone may be able to receive data only as quickly as 300bps.

With the throughput, display, and computational limitations of handheld devices, running a browser that interprets and displays complex HTML information is out of the question. The alternative is to create a new, scaled down, highly structured language and protocol for limited capability wireless devices and still enable Web access. That’s the idea behind WAP.

**WAP architecture fundamentals**

The WAP model is based on a scaled down version of the standard Web model. The WAP model is consistent with most Web architectural conventions, such as standard naming via URLs and standard protocols. Content typing and content formats are consistent with Web content typing and format. This means that WAP applications and types, such as lists, images, and display formats, are direct analogs to Web applications.

A *card* is the fundamental WAP display unit. A card represents one screen’s worth of data. The basic data can be text, a form, a single-bit bitmap, or a menu. There can be one or more cards per WAP client application. Multiple cards reside in a deck. A WAP browser, or user agent, decides how the information in a card is rendered on its particular display. The rendering and application functionality resides in the top-level application layer called the Wireless Application Environment (WAE). The WAE encompasses a micro-browser, supports code written in WML and WMLScript, and also includes the Wireless Telephony Application (WTA). The WTA contains direct access to telephone related functions.

Instead of calling the WAP browser a micro-browser or client program, the WAP documentation calls it a User Agent.

**WML**

The Wireless Markup Language (WML) is derived from the eXtensible Markup Language (XML). XML is itself a markup language derived from the Standard Generalized Markup Language (SGML), best known as the daddy of HTML.

XML allows designers to custom design document types. Whereas everyone developing in HTML is burdened with its all-inclusive nature, XML can be used to create any type of document. HTML must be all things to all application domains. Not so with XML. Custom languages for specific application domains can be created using XML. Rather than accommodating everyone, an XML-based markup language can service a specific domain, such as limited resource wireless devices. This reasoning prevailed when the decision was made to base WML on XML.

WML is designed with small in mind: a small display, a small amount of memory in the target device, and a small amount of computing power. WML applications are composed of cards that reside in *decks*. A deck can contain multiple cards. A card contains *units of user interaction*. These units of interaction are simple menu selections, text entry fields, text formats, and the like. WML also provides an event-handling capability that is used to navigate from card to card.
Every language has a syntax and syntax elements. WML specifies several syntactic units.

### WML structure and transactions

WML transmissions usually originate at the WAP client, or user agent. The user agent makes a request via binary WML byte codes. The request is typically a URL. A WAP proxy takes the binary byte codes and uncompresses them. The uncompressed WML request is then translated to an HTTP request. The proxy issues the request to a Web server via uncompressed HTTP. Content is accessed at the server, returning HTML or other data accessed on the Web server via CGI scripts. The content delivered from the server is HTML or WML, if the Web server is WML-savvy.

At the WAP proxy, the incoming HTML is converted to WML by an HTML filter, then compressed into binary byte codes. If the incoming content from the Web server is already in WML, only compression takes place. The content is delivered to the WAP client and the user agent displays the new content, which is usually a new card. The entire transaction sequence is shown in Figure 1. With all this activity and the inherently slow data rate of wireless devices, one comes to appreciate the minimal nature and design of WML.

### WML syntax elements

Every language has a syntax and syntax elements. WML specifies several syntactic units. These are entities, elements, attributes, comments, and variables. An entity is defined as a self contained unit. WML contains two types of entities: numbers and characters. Elements define all deck information, and are delimited by tags. Tags are similar to tags in HTML. The structure is `<tag> information_content </tag>`. If there is no content, then the tag form is `<tag/>`.

Attributes are values associated with tags. WML tags have default values. If additional or specific information must be associated with a tag, then an attribute value needs to be specified within a tag. The general form of an attribute within a tag is `<tag attr="some_value"/>`. The important things to watch for is the lack of white space between the attr keyword and the equal sign. The value associated with attr must be enclosed in double quotes.

Comments are identified by parsing an exclamation point and two minus signs as the first characters after a left angle bracket. Comments are ended by two minus signs and a right angle bracket. The form is `<!-- comments go here -->`. WML also allows variables in cards and decks. Variables are identified by a single dollar sign, $, followed by the variable name. Variables can be of the form $identifier, $identifier, or $identifier:conversion.

### A simple WML deck

One of the simplest forms for a WML deck is a single card with a text prompt:

```
<wml>
  <card>
    <p>Working with WAP</p>
  </card>
</wml>
```

A WML deck is enclosed by the tags `<wml>` and `</wml>`. This denotes the beginning and ending of a deck, while `<card>` and `</card>` denote the beginning and ending of a card. The `<p>` and `</p>` denote a paragraph, and the enclosed text, “Working with WAP,” is displayed.

This is an entirely static example deck, with no provision for user input, and no soft keys specified. The most common construct for assigning soft keys and taking action is the `<do>` element. A typical `<do>` statement could be `<do type="ACCEPT" label="Next">`. This tag, like the others, must be terminated, in this case, with a `</do>`. The type=”ACCEPT” part of the `<do>` statement indicates an action type. The
Like any language, WML has syntax rules and requires a parser to determine if a submitted program is correct.

**Listing 1** Two card example navigating with <do>

```xml
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card1>
    <do type="ACCEPT" label="Next">
      <go href="#card2" />
    </do>
    <p>
      Working with WAP
    </p>
  </card1>
  <!-- Card number 2 follows -->
  <card2>
    <do type="ACCEPT" label="Last">
      <go href="#card1" />
    </do>
    <p>
      Working with More WAP
    </p>
  </card2>
</wml>
```

This rather cryptic statement says the WML document is based on XML version 1.0, and the version of WML used is 1.1 and can be found at the URL indicated.

**WML navigation and events**

Navigation in WML is tied closely to events and tasks. Events are usually bound to tasks and result in navigating to a new URL. Events can be intrinsic or explicit. Intrinsic events enable WML elements to generate events when interacting with a user. If a particular type of action occurs, the event is generated. Intrinsic events are recognized by four WML elements: ontimer, onenterforward, onenterbackward, and onpick. These are types associated with an onevent element. For example, an onenterforward event is generated when a new card is entered. Explicit events are generated by constructs such as <do> statements, as explained previously.

**Tasks**

WML supports four major “inter-card traversal” mechanisms, or tasks. The tasks are go, prev, noop, and refresh. The go task can take up to 10 steps and finally resolves to navigating to a URL. The prev task can take up to seven steps and results in popping a URL off the history stack and locating a card assigned to the URL. The noop task does no processing. The refresh operation redispers the current card. This operation is used if any variable’s state changes affect the display and need to be displayed. If any of the tasks fail to execute, the currently displayed card is maintained without any changes to variables, bindings, or state.

**WMLScript**

WML, like HTML, is static. WMLScript, like JavaScript, adds dynamic capabilities to WML.
WMLScript adds programming language constructs such as if-then statements. WML is weakly typed, and it’s easy to lose track of variables between native WML and WMLScript.

WMLScript looks much more like a conventional programming language than WML. Since WML is a static language, some serious issues come up, such as user data input validity testing, target device resource access, and local message generation to users. How does WML know if a user entered a name instead of a phone number? What if the user enters something wrong and an error message must be displayed to alert the user? Does this mean that a new card must be accessed through a WAP proxy and Web server just to get a simple error message? WMLScript addresses these problems and offers enough high level language features to solve them.

WMLScript is function-oriented. There is no call to main(), as in a C program. All functions are stored in compilation units and stored on a server for execution. The forms of WMLScript functions are shown in Listing 2.

WMLScript functions follow specific rules and have rigid constraints, such as always returning a value, passing parameters by value only, and not allowing function nesting. WMLScript supports two types of functions: external and internal. External functions are declared when the function is used outside of its compilation unit. This is like making a class member public in C++. If a function is used strictly withing its local compilation unit, then the function is declared normally and is not visible to the outside world. A block of code is like a block of code in C or C++.

Listing 3 shows how to link WMLScript functions to a WML page. This example uses the WMLScript library function setVar() to set a variable in a WML program. The example displays the number 5 on the client’s display. The function GetNum() is called from within the <go> element in Card1. The WML program stops execution, GetNum() is referenced and interpreted, and the results are returned in the WML variable NUM in Card2. The WMLScript function uses

### Listing 2: WMLScript function formats

```plaintext
// for externally accessible functions
extern function FunctionID(params)
{
    BlockOfCode;
};

// for local calls only
function FunctionID(params)
{
    BlockOfCode;
};
```

### Listing 3: WML/WMLScript variable interaction

```plaintext
<wml>
<!-- This file is named num.wml -->
<!-- Card1 calls GetNum() -->
<card1>
    <p>
        <do type="accept" label="Get">
            <go href="num.wmls#GetNum()"/> <!-- call GetNum() -->
        </do>
    </p>
</card1>

<!-- Card2 displays result of GetNum() -->
<card2>
    Num Is: $(NUM)
</card2>
</wml>

// This function is in num.wmls
extern function GetNum()
{
    var number = 5;
    WMLBrowser.setVar("NUM", number);
    WMLBrowser.go("num.wmls#card2");
};
```
The WAP architecture is built on a six-layer protocol based on the Open Systems Interconnect (OSI) model.

The Lang library is a catch-all source of functions that are related to core WMLScript. The Lang functions range from determining the absolute value of a numerical parameter to determining the character set supported by a WMLScript interpreter. The Float library supports floating-point functions typically used in an application, such as floor() and ceil(). The String library contains string manipulation functions such as compare() and length(). The String library also contains some handy instructions such as find(), which locates a substring, squeeze(), which removes extra white space between strings, and trim(), which eliminates excess leading and trailing white space in a string.

The URL library contains functions that manipulate URLs. The URL syntax supported is as follows:

```
<scheme>://<host>:<port>/<path>;<params>?<query>#<fragment>
```

The URL library functions range from checking the validity of a URL to resolving absolute URLs. The majority of the URL functions are prefixed by get, which returns URL content and string information. Functions such as getParameters() and getPort() return the parameter list used in a URL and the port number specified in a URL, respectively.

The WMLBrowser library contains the functions that link to WML variables. Important functions such as getVar() and setVar() retrieve and set variables in a WML program. Other functions in the WMLBrowser library have the same name as base WML functions, such as go() and prev(). This duplicate naming can be a source of confusion if an author isn’t careful.

Underneath the Session Layer is the Transaction Layer, which contains the Wireless Transaction Protocol, or WTP. The WTP is designed with thin clients in mind. The Transaction Layer supports three transaction classes: unreliable one-way requests, reliable one-way requests, and reliable two-way request and reply transactions.

The next two layers down the stack are the Security and Transport Layers. The security protocol is based on the WAP architecture is built on a six-layer protocol based on the Open Systems Interconnect (OSI) model.
The Web is now part of our daily lives and culture, but it is still somewhat chained to the desktop. WAP is designed to break that chain.

on the Secure Sockets Layer (SSL) protocol. Data integrity is ensured at this layer, and data is encrypted using a private key algorithm. The Transport Layer contains the WDP, or Wireless Datagram Protocol. This layer provides the WAP equivalent of TCP and UDP, with the emphasis on connectionless datagrams. The Transport Layer allows the upper layers to operate transparently to the data requirements of the bearer services. On the bottom of the stack are the Bearer Services. Bearers are the digital phones and the data services they provide.

### Developing WAP applications

There are free WML and WMLScript development tools available on the Web. Two are currently available from Phone.com (formerly Unwired Planet) and Nokia. The full WML specification set is available from the WAP Forum. The Phone.com tools require a connection to a server, but this can be localized by configuring a server on the local machine the development tools are downloaded on. Most developers use the free Apache server and configure it locally. The Nokia toolset doesn’t require an external server, although you can use one if you like. The Nokia development kit requires the Java runtime library, and recommends 64MB of RAM.

Both development toolsets have well-thought-out user interfaces, and each has a virtual phone handset on the right of the screen where you can see your WML/WMLScript application in action.

### Go forth

The Web is now part of our daily lives and culture, but it is still somewhat chained to the desktop. WAP is designed to break that chain. Embedded systems programmers have a great opportunity to create WAP applications for wireless and wired platforms yet to be imagined.

---

**FIGURE 2** The WAP six-layer protocol stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Layer</strong>&lt;br&gt;(WAE or wireless application environment)</td>
<td>Contains User Agents, WML, WMLScript, and WTA applications.</td>
</tr>
<tr>
<td><strong>Session Layer</strong>&lt;br&gt;(WSP or wireless session protocol)</td>
<td>Supports two services: connectionless and connection-oriented.</td>
</tr>
<tr>
<td><strong>Transaction Layer</strong>&lt;br&gt;(WTP or wireless transaction protocol)</td>
<td>Supports three classes of transactions: unreliable one-way requests, reliable one-way requests, and reliable two-way requests.</td>
</tr>
<tr>
<td><strong>Security Layer</strong>&lt;br&gt;(WTLS or wireless transport layer security)</td>
<td>Provides session-layer data encryption.</td>
</tr>
<tr>
<td><strong>Transaction Layer</strong>&lt;br&gt;(WDP or wireless datagram protocol)</td>
<td>Supports WAP version of TCP or UDP. Responsible for packet fragmentation, reassembly, and maintenance of end-to-end service.</td>
</tr>
<tr>
<td><strong>Bearers</strong>&lt;br&gt;GSM, CDMA, CDPD</td>
<td>This is the physical interface to telephony and wireless hardware and their services.</td>
</tr>
</tbody>
</table>

**Resources**

- **WAP Forum.** This is the WAP mother ship. The full set of WAP technical specifications are found on the WAP technical link: www.wapforum.org.
- **Nokia Wireless Data Forum.** This offers a WAP server trial kit and the Nokia WAP Toolkit along with a well written documentation set that’s loaded with good examples. The WAP Toolkit is free but requires developer registration: www.forum.nokia.com/developers/wap/.
- **Phone.com.** Go to the Developer Web Site to download the development kit. Like Nokia, the toolkit is free but requires registration: www.phone.com/developers/index.html.

---

Jeff Stefan is a system engineer at OnStar. He has worked in embedded systems software for many years, is a frequent speaker at the Embedded Systems Conferences, and is the author of over 15 software articles. He is currently writing his first book, entitled Embedding Artificial Intelligence. His e-mail address is jmstefan@ mindspring.com.