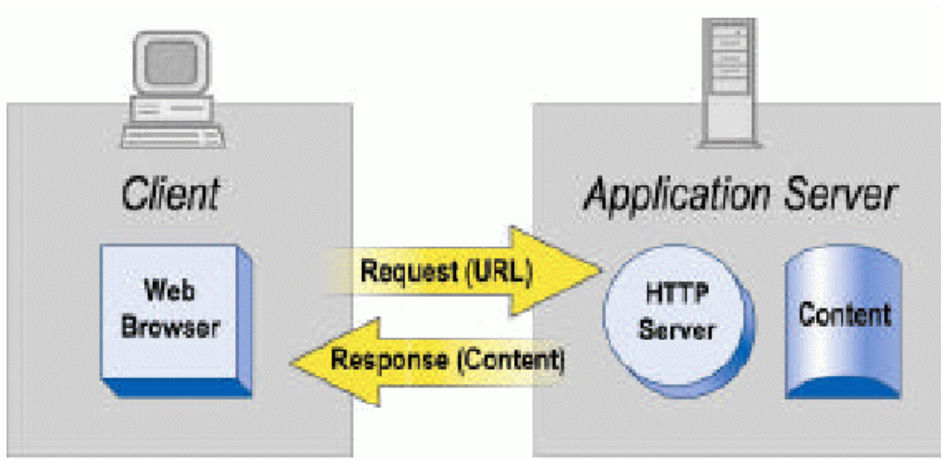
WAP : Wireless Application Protocol

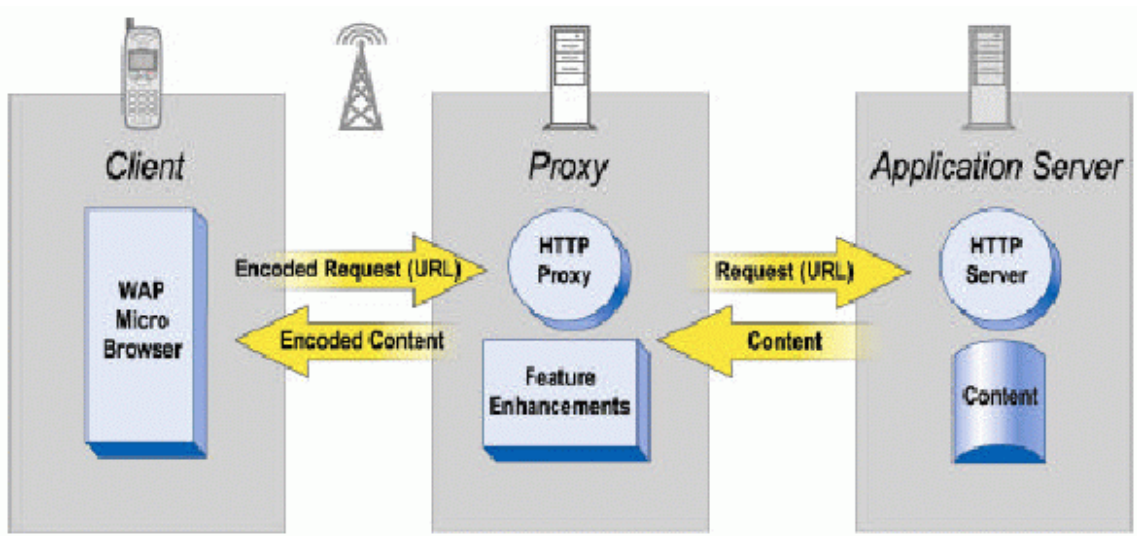
- a standard developed by the WAP Forum :
  - Nokia, Ericsson, Motorola et al.
  - www.wapforum.org – now consolidated into the *Open Mobile Alliance*
- Internet services for users of mobile devices
- to be accessible over *bearer* services on all standard cellular systems
  - GSM, TDMA & CDMA (USA), GPRS, 3G systems etc.
    - » e.g. the GSM CSD (Circuit Switched Data) bearer service
      - the most widely used GSM data service providing a data rate of 9.6kbs
      - with error correction and flow control
- defines a layered communications protocol
  - including an application environment
  - using a client-server approach
- incorporates a simple *microbrowser* into the mobile device
  - only requires the limited resources a mobile can provide
  - puts intelligence into *WAP gateways* rather than the mobile

- Wireless Application Environment (WAE)
  - – includes a microbrowser for the WAP Markup Language WML
  - – WML better adapted to mobile device constraints than HTML
    - » has to cope with small screen sizes and low data rate
    - » less concerned with visual aspects of document rendering
      - - and interactive elements e.g. buttons
    - » more concerned with functionality
      - - microbrowser can choose its own appropriate representation of the WML
    - » XML compliant with its own DTD Document Type Definition
  - – also includes a micro Virtual Machine
    - » for the microbrowser's scripting language, WMLScript, to execute in
    - » suited to the memory and CPU constraints of a mobile device
  - – WMLScript derived from ECMAScript – a standard scripting language
    - » as is Javascript
  - – support for standard content types, including vCard and vCalendar
    - » used for communicating phonebook and calendar data between applications
  - – the top level of the WAP protocol stack

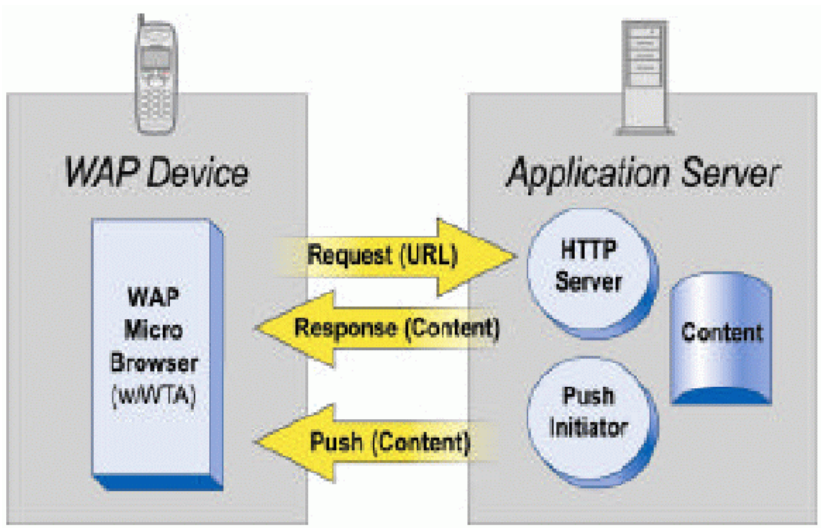
- Architecture
  - World Wide Web :



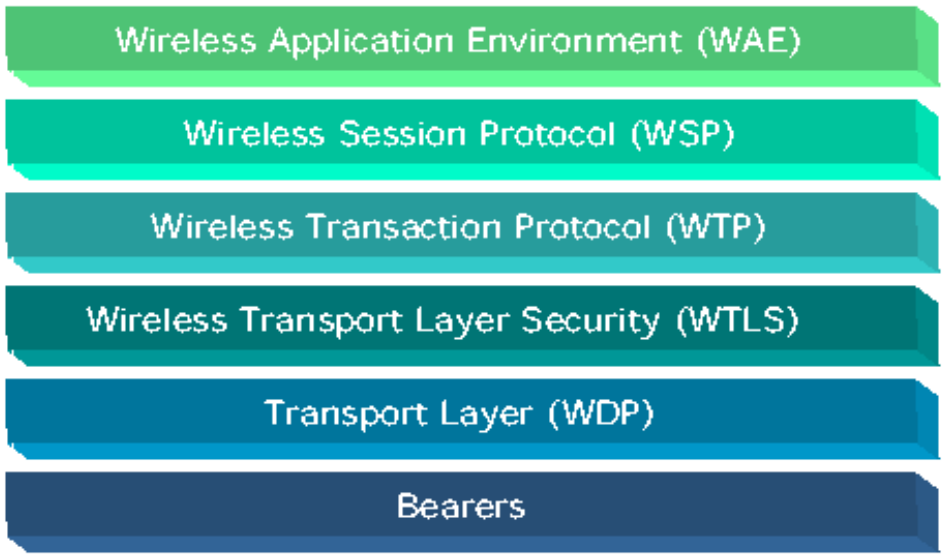
  - WAP model :



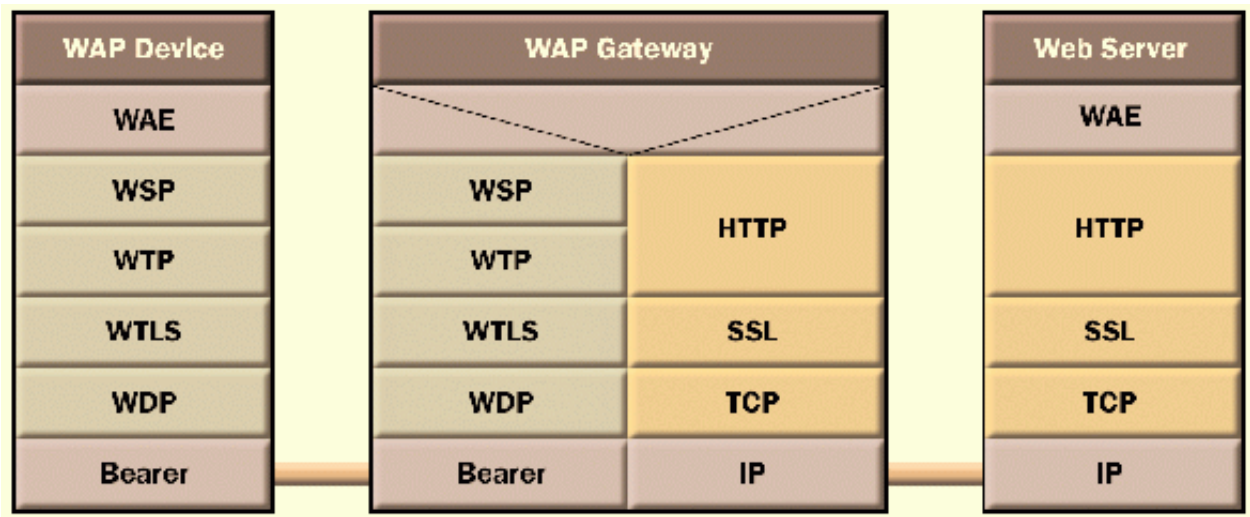
- interposed proxy gateway :
  - – protocol adapters to translate between WAP protocols and Web protocols used to access network services
    - » e.g. HTML to WML
  - – encoding and compilation of WML and WMLScript from text to binary
  - – binary format minimises the amount of data to be sent over the network
    - » much more efficient than HTTP servers for PCs
  - – reduces the amount of processing that the mobile needs to perform
- *push* technology also incorporated :
  - – delivery of content without previous user interaction
  - – a WAP session opened automatically by the network
  - – e.g. stock market prices, SMS-type messages but longer



- WAP 1.0 protocol stack :



- gateway configuration :

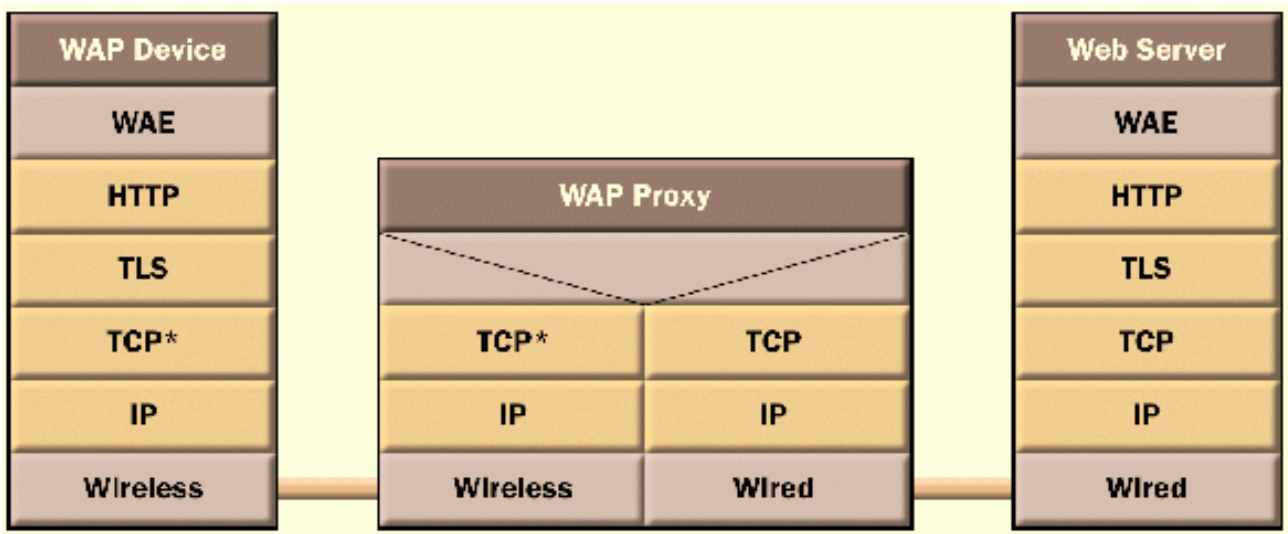


- Wireless Session protocol
  - – responsible for managing the session between mobile and server
  - – supports a connection-oriented service above the WTP transaction protocol
  - – and a connectionless services operating above the WDP datagram protocol
    - » can be secure or non-secure
  - – includes the ability to suspend and resume sessions
    - » important because connections may be dropped and need reconnecting later to continue the session from where it left off
      - e.g. out of reception area, entering a railway tunnel etc.
    - » support for long-lived sessions
  - – session setup involves negotiation of the nature and capability of the mobile and the server
    - » not needed for connectionless service
  - – transmits session and content headers
    - » define character sets, language, mobile device profile
  - – handles coding and decoding of content

- Wireless Transaction protocol
  - provides a request/reply based reliable transport mechanism :
    - » selective retransmission of lost or corrupted messages
    - » message concatenation of PDUs into transport SDUs
    - » 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
  - three classes of transport defined :
    - » class 0 : an unreliable, stateless connection like UDP
      - no result is communicated and abort function not supported
    - » class 1 : a reliable stateful transport
      - supports retransmission but does not transmit a result; abort supported
    - » class 2 : reliable and stateful
      - returns a result for every message sent
      - also supports a *hold on* reply as a result which indicates that the server is busy processing the request

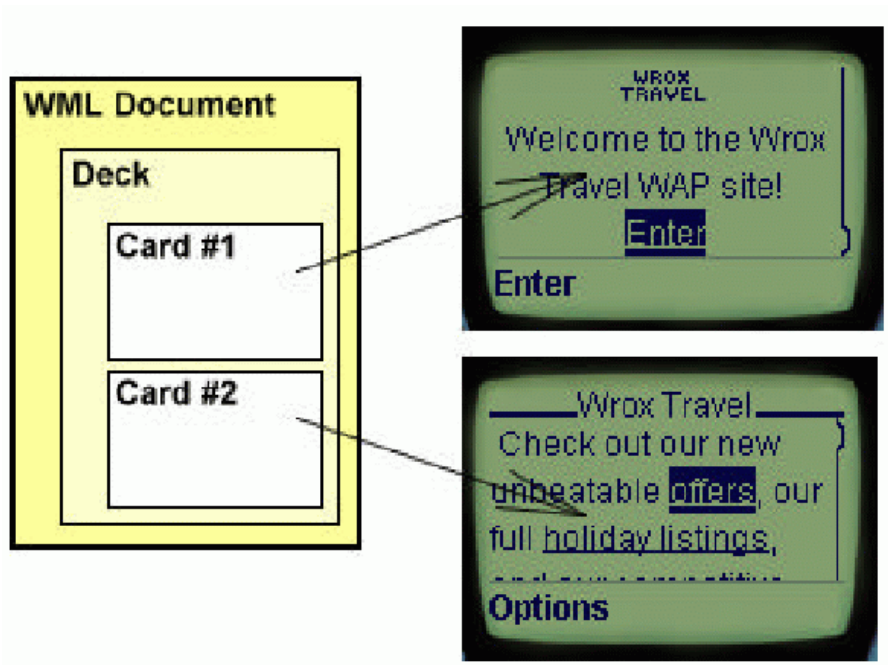
- Wireless Transport Layer Security
  - provides secure transmission of data across the air network
  - based on SSL, Secure Sockets Level protocol
  - supports privacy, message integrity and authentication
  - includes key exchange methods, certificates, symmetric and asymmetric encryption algorithms and signature functions
  - has been criticised for allowing weak encryption algorithms
- Transport Layer
  - preferred transport layer is UDP over IP but many mobile networks do not support this
  - the Wireless Datagram protocol used instead
  - provides a connectionless non-reliable transport protocol
  - message segmentation and reassembly
  - UDP-type port numbers
  - uses a Wireless Control Message Protocol, similar to ICMP, to signal error conditions

- WAP 2.0 standard
  - convergence with the Internet
  - motivated by emergence of high-speed networks, 3G etc. that provide IP support directly to the mobile



  - adds other enhanced features :
    - » enhanced push functionality and user agent profiles
    - » tools for advanced telephony applications
    - » external functionality interface for applications outside the defined WAE capabilities

- WML : WAP Markup Language
  - based on a *deck of cards* metaphor
    - » a document is analogous to a deck
    - » a card is equivalent to an individual screen or unit of display



  - a deck is the *unit of transmission* between gateway and mobile
  - focuses on semantics of an element rather than rendering
    - » allows implementation of rendering to be tailored to capabilities of the mobile

– supports text and images, user interaction, navigation, variables etc.

– layout and presentation *hints* can be given

– *templates* to specify characteristics to be applied to all cards in the deck

– navigation between cards in a deck and between decks

» through URL hyperlinks or history

» a URL taken to be the first card in a deck

- other cards can be referenced using *fragment anchors*

– tasks bound to events and triggered by user interaction

» `onenterbackward`, `onenterforward` events for navigating back and forth

» `onpick` event triggered when user picks an item from a list

» `ontimer` causes a jump to a URL when a timer expires etc.

– tasks associated with WML *elements*

» e.g. a `<go>` element indicates navigation to a URI

» a `<go>` element in an `<onevent>` of type `oneventforward` in a `<card>` element binds the `<go>` element to the `oneventforward` for that card

– microbrowser maintains a context that can contain variables
  » variable names start with a `$`
– variable substitution allows for cards to be customised on the mobile
– set by a `<setvar>` element, an `<input>` element or a `<select>` element
  » an `<input>` element specifies a text entry object
  » a `<select>` element gives a choice from a list of options, possibly hierarchically
– a `<timer>` element is used to set the value of a timer on entry to a card
  » in tenths of a second
– links specified with `<anchor>` or `<a>`
– tables specified with `<table>`, `<tr>` and `<td>` elements
– etc.

- Examples of WML
  - the document prologue :

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

  - defining a template :

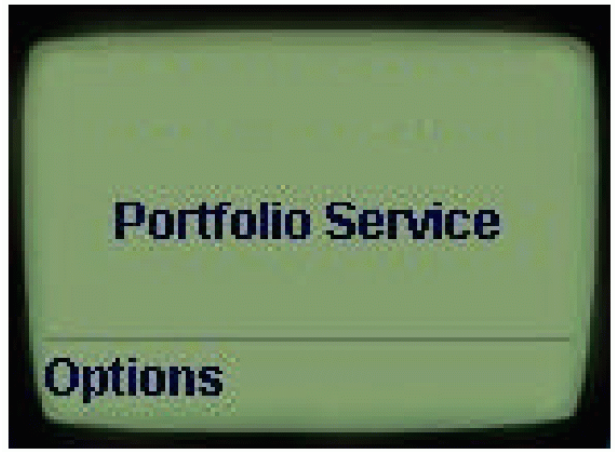
```
<wml>
  <template>
    <do type="goBack" name="goBack"
      label="Back"> <prev/>
    </do>
  </template>
  <card id="init" newcontext="true">
  </card>
</wml>
```

» results in the Back option being displayed on each card in the deck

» the <do> element associates the label Back with the <prev> task

– defining a card :

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



– using anchors :

```
<wml>
  <card id="MainMenu" title="MainMenu">
    <p>
      <anchor>1. Market Indices
        <go href="#MarketIndices"/>
      </anchor><br/>
      <anchor>2. Portfolio Valuation
        <go href="#Portfolio Valuation"/>
      </anchor><br/>
      . . . .
    <p/>
  </card>
</wml>
```

– 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>
```

- an animation example using WML and WMLScript :

```
<?xml version="1.0"?>
   <!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
      "http://www.phone.com/dtd/wml11.dtd">
<wml>
   <card title="A simple animation">
      <onevent type="onenterforward">
         <refresh>
            <setvar name="image"    value="animate1.bmp"/>
            <setvar name="napTime"  value="20"/>
            <setvar name="snore"    value="Zz"/>
         </refresh>
      </onevent>
      <onevent type="ontimer">
         <go href="animate.wmls#main()"/>
      </onevent>
      <timer value="$(napTime)"/>
      <do type="accept" label="$(snore)">
         <go href="device:home"/>
      </do>
      <p align="center">
         <img alt="" src="$(image)"/>
      </p>
   </card>
</wml>
```

```
extern function main()  {
    var image   = WMLBrowser.getVar("image");
    var curTime = WMLBrowser.getVar("napTime");
    var napTime = Lang.parseInt(curTime);

    Console.println("***********Animation Vars***********");
    Console.println("Image: " + image);
    Console.println("Nap Time: " + napTime);
    //reduce napTime; at zero, animation is complete
    napTime = napTime - 2;
    WMLBrowser.setVar("napTime", napTime);
    if (napTime > 0) {
        if (image == "animate1.bmp") {
            WMLBrowser.setVar("image", "animate2.bmp");
            WMLBrowser.setVar("snore", "zZ");
        } //if animate1.bmp
        else {
            WMLBrowser.setVar("image", "animate1.bmp");
            WMLBrowser.setVar("snore", "Zz");
        } //else
        WMLBrowser.refresh();
    } //if napTime > 0
    else {
        WMLBrowser.setVar("image", "animate3.bmp");
        WMLBrowser.setVar("snore", "Hhhhnnnyyy?");
        WMLBrowser.refresh();
    }
} //main()
```

The file animate.wml consists of a single card. When the user selects the Run Animation option, the WML sets the variable time to a beginning value of 20 seconds and sets the initial image file value:

```
<onevent type="onenterforward">
    <refresh>
        <setvar name="image"    value="animate1.bmp"/>
        <setvar name="napTime"  value="20"/>
        <setvar name="snore"    value="Zz"/>
    </refresh>
</onevent>
```

After 20 seconds, the ontimer event fires and the WML calls the main() function in animated.wmls:

```
<onevent type="ontimer">
    <go href="animated.wmls#main()">
```

The main() function reduces the time variable by 2 seconds, swaps the image file, and returns the new time and image file variable values to the UP.Browser. The card then displays the new image:

```
<p align="center">
    <img alt="" src="$(image)"/>
</p>
```

The process is repeated each time the ontimer event fires until the time variable's value drops to zero.

The image and time processing take place in animated.wmls which defines one external function:

```
extern function main()
```

The main function initializes its own variables to the values the WML card set using the WMLBrowser library function getVar:

```
var image   = WMLBrowser.getVar("image");
var curTime = WMLBrowser.getVar("napTime");
var napTime = Lang.parseInt(curTime);
```

The last line in this block converts the string value derived from time to an integer value that the main function can use to alter the value of the variable.

The next block reduces the value of remainingTime by 2 and assigns the new value to the WML variable time using the WMLBrowser.setVar function. This sequence controls both the interval between image file swapping and ensures that the animation stops when remainingTime drops to zero.

```
//reduce napTime; at zero, animation is complete
    napTime = napTime - 2;
    WMLBrowser.setVar("napTime", napTime);
```

The last block tests remainingTime for a value greater than zero. If remainingTime is greater than zero, the next few lines test the current value of the image variable, swap the bitmap file accordingly, and set the image variable to the new value using the WMLBrowser.setVar function. After the image file swap is complete, WMLBrowser.refresh function refreshes the variables, causing the user's display to be updated with the new image file and also updating the time variable.

```
if (napTime > 0) {
    if (image == "animate1.bmp") {
        WMLBrowser.setVar("image", "animate2.bmp");
        WMLBrowser.setVar("snore", "zZ");
    } //if animate1.bmp
    else {
        WMLBrowser.setVar("image", "animate1.bmp");
        WMLBrowser.setVar("snore", "Zz");
    } //else
    WMLBrowser.refresh();
} //if napTime > 0
```

When remainingTime drops to zero, no processing occurs and the timer loop ends.