Weekend Wiki
— or how to get organized

Redondo Beach
18 April 2005

Mike Cowlishaw
IBM Fellow
Overview

• What’s a Wiki?
• How it works (mechanics and code)
• Extra features
• Possible enhancements
What’s a Wiki

• From “wiki wiki” — Hawaiian for *quick*
  – Ward Cunningham, 1995

• Allows the creation and editing of web pages using only a browser

• Makes it easy to add links between pages

• Has *shorthand* for markup
Wiki markup

• Varies by implementation. Generally has structural markup (lists, etc.) and ‘inline’ markup (italic and bold emphasis, etc.)

Example:

* Bulleted list, ‘‘italics’’
* ‘’’Bold item’’’ -- 10^6
* And a link to [Another Page]
Markup result

* Bulleted list, ""italics"
* """"Bold item"""" -- $10^6$
* And a link to [Another Page]

- Bulleted list, *italics*
- **Bold item** — 10^6
- And a link to [Another Page]

(more examples in a moment)
How does a Wiki work?

Browser

Web server

May be on the same computer
Demonstration
Why did I write my own?

• Most Wikis are built on databases; I wanted to use plain text files, and integrate Wiki pages with other files, references, etc.

• I wanted a common Wiki markup for web pages, my notes, and Wikipedia superset
  – allows off-line Wikipedia edit and preview

• I needed to cover multiple projects, with easy interlinking
Why did I write my own?  [2]

• I wanted it to be really fast for core features (pure HTML, no images, no JavaScript)

• I wanted to make ‘publishable’ static snapshots (no need for a web server); see:

   http://www.cary.demon.co.uk/memowiki/

• I was in a Rexx programming mood …
... on the same computer (address is http://127.0.0.1)
and the server can be written in Rexx ...
A Rexx Web server

socket=SockSocket("AF_INET", "SOCK_STREAM", "IPPROTO_TCP")
call SockSetSockOpt socket, "SOL_SOCKET", "SO_REUSEADDR", 1
address.!family="AF_INET"
address.!port=80 -- HTTP well-known port
address.!addr="INADDR_ANY"
rc=SockBind(socket, "address.!")

signal on halt name halted
client='?'
do forever -- handle each request

/* ---- see next slide ---- */

do forever

end -- forever loop

halted: -- here on Halt break
if client\='?' then call SockClose client
call SockClose socket
Incoming HTTP data stream

Request verb, selector, version

Headers

POST /Cognition/Glossary HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: 127.0.0.1:8087

textinput=abcdef&action=edit

Empty line

Data (body of request)
Web server inner loop

do forever
    rc=SockListen(socket, 1)    -- wait for a client
    client=SockAccept(socket, "client.!" )
    data=''
    bytes=SockRecv(client, "data", 1000)
    heads=0
    do forever                  -- get header lines
        parse var data header.heads ('0d0a'x) data
        say heads':' header.heads
        if header.heads=='' then leave -- reached separator line
        heads=heads+1
    end
    say 'data: "'data"'        -- body remains in data
    reply='&lt;html&gt;Hello, in &lt;b&gt;bold&lt;/b&gt;.&lt;/html&gt;'
    rc=SockSend(client, reply, length(reply))
    call SockClose client       -- done with the connection
    client='?'                  -- no active client
end -- forever loop
/* Minimal single-thread HTTP server in Rexx. MFC 2005. */
-- Load all functions
call rxfuncadd sysloadfuncs, rexxutil, sysloadfuncs
call sysloadfuncs
call rxfuncadd "sockloadfuncs", "rxSock", "sockloadfuncs"
call SockLoadFuncs('quiet')
CRLF='0d0a'x        -- useful
/* Get a socket... */
socket=SockSocket("AF_INET", "SOCK_STREAM", "IPPROTO_TCP")
if socket<0 then do
  say 'SockSocket failed:' socket
  return socket
end
call SockSetSockOpt socket, "SOL_SOCKET", "SO_REUSEADDR", 1
/* Bind... */
address.!family="AF_INET"
address.!port=80              -- HTTP well-known port
address.!addr="INADDR_ANY"
rc=SockBind(socket, "address.!")
if rc<0 then do
  say 'SockBind failed, errno='SockSock_Errno()
  return rc
end
client='?'
signal on halt name halted
say 'Listening...'
/* Main loop waiting for requests */
do forever
  /* Listen for a request... */
  rc=SockListen(socket, 1)
  if rc\=0 then do
    say 'SockListen error, rc='rc
    return rc
  end
  /* Accept a request... */
  client=SockAccept(socket, "client.!")
  if client\=l then do
    say 'SockAccept() failed'
    return -1
  end
  /* Receive the message... */
data=''
  bytes=SockRecv(client, "data", 1000)
  if bytes<0 then do
    say 'SockRecv() failed'
    return -1
  end
  say client.!family client.!port client.!addr', got: ' bytes
  -- put header lines into HEADER., with the request in HEADER.0
  heads=0
  do forever
    parse var data header.heads (crlf) data
    say heads': ' header.heads
    if header.heads=='' then leave -- reached separator line
    heads=heads+1
  end
  -- here the body of the request (if any) remains in DATA
  say 'data: ''data'''
  /* Send a reply... */
  reply=<'html>''<b>This is a <b>bold</b> statement.</b>''crlf,
           '<b></b>'crlf,
           '</html>'
  rc=SockSend(client, reply, length(reply))
call SockClose client  -- done with the connection
  client='?'             -- no active client
end -- forever loop
halted: -- here on Halt break
/* Close sockets and TCP/IP */
if client\='?' then call SockClose client
call SockClose socket
call SockDropFuncs
exit
How is the Wiki programmed?

• Server receives URL (selector):
  /MemoWiki/Markup

  Project name

  Page name

• May have action, too (up to programmer):
  /MemoWiki/Markup?edit
Project and Page names

• Project name is name of a directory (in the MemoWiki data tree)

• Page name is name of a .wiki file stored in the wiki subdirectory (or a .ref file in the refs subdirectory)

\d:\\wikiroot\\MemoWiki\\wiki\\Markup.wiki
Safe names

- URLs allow only the characters A-Z, a-z, 0-9, + − * / . _ @ (and + / @ are reserved)
  - cannot really use * and . in file names
  - which leaves only −, _, and alphanumerics

- Use _ for blanks, − for escapes, e.g.:

  My_Page    To-2Ddo_ma-A4ana

(To-do mañana)
/* ------------------------------------------------------------------ */
/* safe2user - Return human-readable name from safe file/url */
/* */
/* * Arg1 is a safe file/url name */
/* */
/* * See user2safe for details. */
/* * ------------------------------------------------------------------ */
parse arg safe
safe=translate(safe, ' ', '_')          -- any blanks
out=""
do forever
  p=pos('-', safe)
  if p=0 then leave                     -- no more escapes
  parse var safe pre =(p) +1 hex +2 safe
  if datatype(hex, 'x') then ins=x2c(hex)   -- valid escape
    else ins='-'hex     -- bad: leave as-is
  out=out||pre||ins
end
return out||safe

/* ------------------------------------------------------------------ */
/* user2safe - Return safe file/url ID given a human-readable name */
/* */
/* * Arg1 is any string */
/* */
/* * URLs allow only the characters + - * / . _ @ and alphanumerics. */
/* */
/* * URIs reserve (of these) + / @ */
/* */
/* * Windows files do not safely allow (of these) * and . */
/* */
/* * Therefore only - and _ are available. The transformation we use */
/* * to generate safe names is therefore: */
/* * Alphanumerics are unchanged */
/* * Blanks -> underscore */
/* * All others -> -xx [hex escape, system page encoding] */
/* * ------------------------------------------------------------------ */
parse arg data
alphanumb=" abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
out=""
do forever                              -- generate escapes
  v=verify(data, alphanumb)
  if v=0 then leave
  parse var data ok =(v) char +1 data
  if v=1 then ok=""
    -- matched end of string
    out=out||ok""c2x(char)
  end
return translate(out||data, '_', ' ')
ViewPage.rex

• Calls Header to write header to output file
  (Markup.html, in \MemoWiki\#cache)
• Calls RenderPage to render page content
  – which calls wiki2html to generate HTML from
    the Markup.wiki source file
• Writes standard footer to output file
• Returns to server, which sends the output
  file to the browser
**EditPage.rex**

- Calls Header to write header to output file
- If previewing, calls RenderPage to render page content
- Adds a text input box with content of the `Markup.wiki` file (form set to use POST)
- Writes standard footer to output file
- Returns to server, which sends the output file to the browser
ReplacePage.rex

- Saves the current `Markup.wiki` file in the archive (`\MemoWiki\#archive`)
- Writes a new `Markup.wiki` file from the data (body of the request) from the browser
- Cleans the archive for the page (optional)
- Returns to server, which redirects the browser to view the page …
Redirecting the browser

- Sends a ‘302’ response:

```html
<!doctype html public "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>Moved</title></head>
<body>
<h2>Document moved...</h2>
<p>This document has moved.</p>
</body></html>
```

(Browsers don’t usually show the HTML)
Directory Structure

• WikiRoot
  – #server        web server and .rex files
  – #temp          for one-off html files
  – MemoWiki       project directory
    – wiki         .wiki files
    – refs         .ref files
    – files        general files
    – #cache       re-usable HTML files
    – #archive     saved (old) pages
Actions (page level)

- View
- Edit
- View printable (no buttons)
- Google (Web or Scholar)
- History of this page
- Links to this page
Actions (project level)

- Go to home page (/Project/Project)
- Search pages
- Add or view page (‘go to page’)
- Add a new reference
- Compact index of pages
- Recently changed pages / pages by date
- Clean up archive
- Build static snapshot
Actions (top level)

- List projects
- Add a new project
- Help (also at lower levels)
  - simply views a page in MemoWiki project
- Explore all
  - also available at project level
- Server maintenance
  - special commands to server
Demonstration
Helper code – num2word

• For better messages, for example:


One reference has its named file missing, indicated by a ‡ mark. Three references have no associated file.
num2word

/* ----------------------------------------------------------- */
/* num2word - number, or word for number if 1-10               */
/*                                                             */
/*   Arg1 is an integer                                        */
/*   Arg2 is 1 if the first letter should be a capital         */
/* ----------------------------------------------------------- */
num2word:
   parse arg num, cap
   if num>10 then return num

   nums='one two three four five six seven eight nine ten'
   text=word(nums, num)
   if cap\=1 then return text
   parse var text c1 +1 rest
   return translate(c1)rest
Wiki2html tips

• Outer loop splits out tags, links, and text

  text <tag> more text [link] remaining text

• Tags are passed straight through to output HTML file

• Links are converted to
  <a href="...">content</a>
Wiki2html tips – text segments

• Text segments are processed by splitting into lines (special characters at start of lines indicate structural markup)

• Structural markup processed as needed (generate headers, lists, etc.)

• Text is finally passed to a ‘textout’ routine, which handles inline markup (italics, etc.)
Wiki2html tips – text output

• Accents: fum~e’ → fum&eacute;; → fumé

• then quotes: ‘single’, “double”, it’s

• superscripts: $10^6$ → 10<sup>6</sup> → $10^6$

• the rest are easy substitutions:

```text=text=changestr('--', text, '&ndash;')```
Other features one could add…

• Compare page versions (‘diff’)

• Multiple (identified) users – probably would be worth adding database for that
  – users’ preferences, contributions, watchlist

• … see other Wikis (e.g., Wikipedia) for more ideas
Questions?

For documentation, see:

http://www.cary.demon.co.uk/memowiki/

(Google: memowiki)