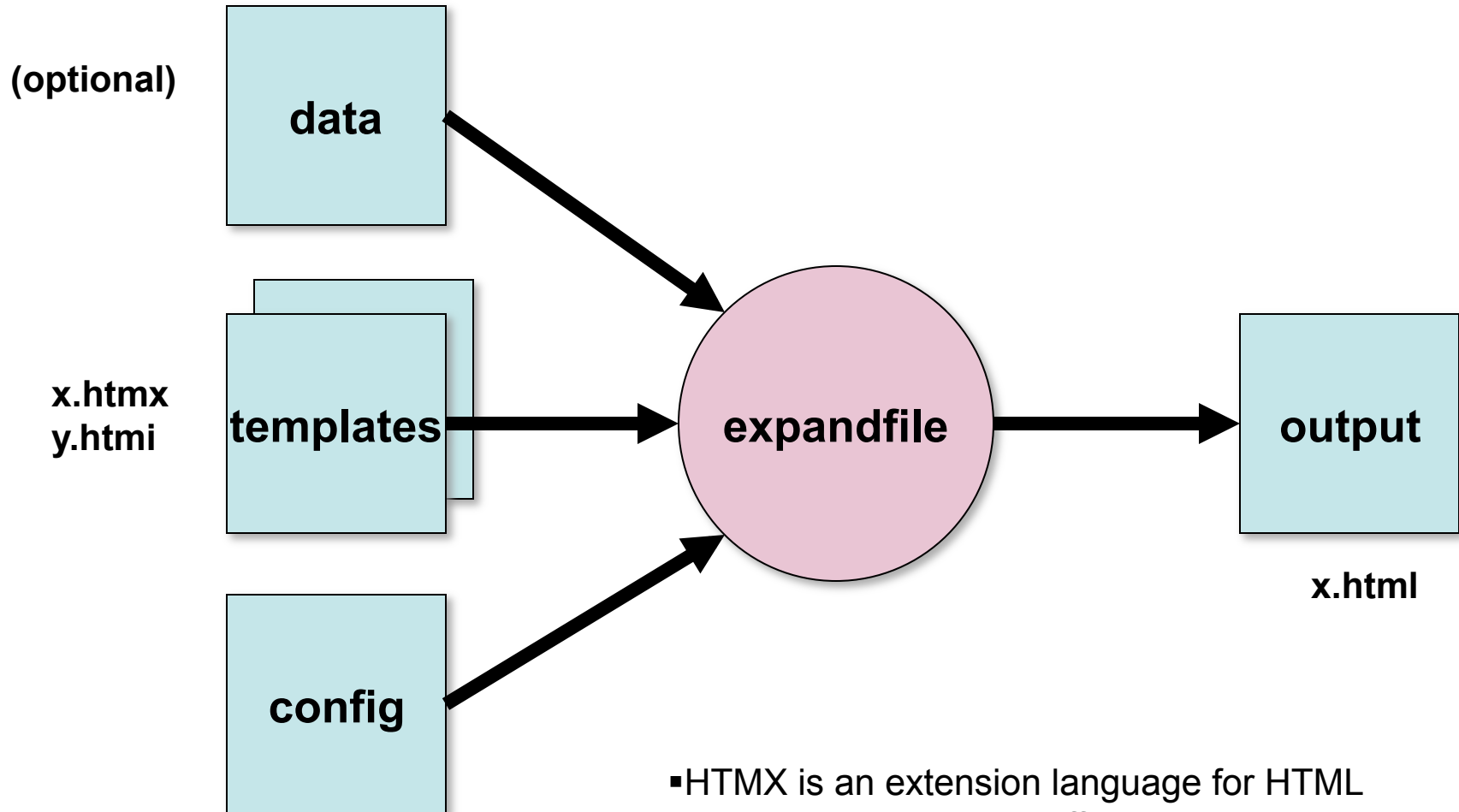


HTMX language

Tom Van Vleck

v25 – 06 Apr 2018

Overview



- HTMX is an extension language for HTML
- **expandfile** can run offline or online
- Can produce any format of output, not just HTML

HTMX Benefits

- **Simplify features of HTML.**
- **Avoid errors.**
- **Edit one file instead of many:**
 - **Standard headers, footers, values.**
 - **One file can depend on another's content.**
 - **One file can depend on another's attributes.**
- **Fill in values automatically.**
- **Can still use any HTML feature.**
- *Pages are statically generated offline.*
- *No security holes introduced.*

HTMX Example 1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML %[htmllevel]%/EN">
<html>
  <head>
    <title>%[titlestring]%/</title>
    %[ *include,=mxstdfmt.html ]%
    %[ *include,=linktags.html ]%
    %[ *include,=hlstyle.html ]%
  </head>
  <body>
    <div class="outer">
      %[ *set,&headingtitle,="Phase One" ]%
      %[ *set,&headingdate,="25 Jul 1995" ]%
      %[ *include,=class2head.html ]%
      <br>
      . . .
    </div>
  </body>
</html>
```

- Contents of `% [...] %` are expanded. The rest is copied.
- Expansions beginning with `*` perform *built-in functions*:
 - Assign `% [*set, &var, value] %`
 - Include file `% [*include, =filename] %`

HTMX Example 2

```
[%[*set,&title,="My Fish Story"]%  
[%[*block,&content,^END]%  
  <p>This is a story called [%[title]%].  
  It will get standard head and foot.  
  </p>  
END  
[%[*include,=story-template.htmli]%
```

Sets variable **title** to the string “My Fish Story”

Sets variable **content** to the block of 3 lines before END.

Reads and expands macro file **story-template.htmli**
which expands the values of variables **title** and **content**
and wraps the content with a standard heading and footer.

Design templates once and use them in many pages.

HTMX Builtins Can...

- **Assign and modify variables.**
- **Include other HTMX files, which may contain variable references or macros.**
- **Execute programs and use their output.**
- **Perform conditional tests.**
- **Write data to files for later inclusion.**
- **Use values from configuration files and environment variables.**
- **Use information from database or file system.**

HTMX example 3

```
1 %[*shell,&humordate,=filemodshort multics-humor.htmx]%  
2 %[*shell,&humorK,=filesizek ../multics/multics-humor.html]%  
  %[*set,&age,=fileagedays ../multics/multics-humor.html]%  
3 %[*set,&humoru,="" ]%  
  %[*if,>,9,age,*set,&humoru,updatedflag]%  
4 <dt><a href="multics-humor.html"><span class="topic">Humor</span></a>  
  %[humoru| |]%</dt>  
5 <dd>Jokes about Multics.  
  <span class="adata">(%[humorK]%K, 1 picture, %[humordate]%)</span>  
</dd>
```

1. Set variable **humordate** to the date a file was modified. (External program **filemodshort** returns the date modified of a file as mm/dd/yy.)
2. Set **humorK** to the size of a file, by calling external program **filesizek**.
3. Set **humoru** to contents of **updatedflag** if file is less than 9 days old.
4. Insert a line beginning with **<dt>**. If **humoru** is nonempty, insert its contents preceded by a blank.
5. Insert a **<dd>** block with the file's size and date last modified.

HTMX example 3a

```
1 [%include,=htmxlib.html]%  
2 [%callv,myfileinfo,"multics-humor"]%  
3 <dt><a href="multics-humor.html">  
4   <span class="topic">Humor</span></a>[%humoru| |]%</dt>  
   <dd>Jokes about Multics. [%fileattrib]%</dd>
```

Use a macro to do the same thing.

1. Include file **htmxlib.html** (once) which defines a set of macros.
2. Call macro **myfileinfo** to get file info and set some variables.
3. If **humoru** is nonempty, insert its contents preceded by a blank.
4. Insert a line with the file size and date modified.

Result: automatic updating of the listing whenever the file changes. Simple source code.

HTMX example 4

```
%[*callv,imgdiv,="sp2.gif",="p2.gif",="horse",="click for larger view.",="pic",=""]%
```

Invoke a macro to generate an image tag, calling a program to get the image size. Output:

```
<div class="pic">  
  <a href="p2.gif"></a>  
</div>
```

*Result: don't need to put image size in your source.
If the image changes, the generated HTML will adjust
when you recompile.*

HTMX Expansion

- `%[var]%` (get the value named `var`)
- `%[varsel | pre | post]%` (general form)
 - Where `varsel` is `var1, var2, var3...`
e.g: `%[title, headingtitle | <h1> | </h1>]%`
 - Returns `pre value post`
 - concatenated with no spaces
 - for the first nonblank value from `var i`
 - looking first in bindings set in the symbol table, then in command environment
 - if all are blank, return nothing.
 - `pre` and `post` may contain `\\n` which will become a newline.

Variables

- **Variables have names**
 - Letters, digits, spaces, ()-+_. are allowed.
 - All-numeric names cause a warning.
- **Variables have string values**
 - Any length.
 - HTMX built-ins may set variable values
`%[*set,&var,="string value"]%`
 - Configuration file may pre-set variable values.
 - Shell environment variables are searched too.
 - Variables can be set to a string returned by an external program.

HTMX Syntax

- Literals begin with =
 - `[%[*set,&x,="abc"]%`
 - Sets `x` to the value `"abc"`, outputs nothing.
 - Quotes are required if you have special characters in the value, like commas, or `[%`. Use the quotes for clarity.
 - Compare `[%[*set,&x,="it is [%[xyz]%"]%`
and `[%[*set,&y,=it is [%[xyz]%"]%`
when variable `xyz` contains `"Tuesday"`.
The first one sets `x` to `"it is [%[xyz]%"` because of quotes
and the second one sets `y` to `"it is Tuesday"`
- The ampersand
 - As in `[%[*set,&x,=abc]%`
 - Indicates that a value is modified by the builtin.
 - If you leave it out, a warning is printed.

HTMX Syntax

- **Nesting**

- Expands inner variable references first.
- `%[*set,&x,=abc%[r]%xyx]%`
- Assuming variable `r` contains "99", this will set `x` to "abc99xyz"

- **Quoting**

- `%[*set,&x,="]%"]%`
- Sets `x` to a right bracket and a percent.
Quotes are interpreted only inside `%[]%`.
Outside of `%[]%` quotes are just characters.

- **Escaping**

- `%[*set,&x,="\ "]%`
- Sets `x` to a quote character. Similarly `\\`, `\%`, etc.
- The `\` character removes the special meaning of the next char.
- `\` is respected everywhere. To input a `\` in text, use `\\` instead.

- **Tracing**

- `%[*set,&_tracebind,="yes"]%`
- Causes `*sqlloop`, `*csvloop`, `*xmlloop`, `*dirloop`, and `*ssvloop` to output a message when they bind a variable.

Builtin Values

(built into `expandfile`)

<code>%[year]%</code>	2004
<code>%[prevyear]%</code>	2003
<code>%[day]%</code>	07
<code>%[month]%</code>	Oct
<code>%[prevmonth]%</code>	Sep
<code>%[monthx]%</code>	10
<code>%[hour]%</code>	09
<code>%[min]%</code>	31
<code>%[date]%</code>	07 Oct 2004
<code>%[timestamp]%</code>	2004-10-07 09:31
<code>%[pct]%</code>	%
<code>%[lbkt]%</code>	[
<code>%[rbkt]%</code>]
<code>%[quote]%</code>	"
<code>%[_currentfilename]%</code>	<code>xyz.htmx</code>

35 HTMX built-in functions

Begin with `*`. Won't cause blank lines in the output if they are the only thing on a line.

- `%[*include,=filename]%`
 - insert the contents of `filename`, expanding variables in it.
 - (use the `*fread2` builtin to read files without expanding values.)
- `%[*set, &varname, value]%`
 - set `varname` to "`value`", for later expansion.
 - `value` can be
 - `name` Value of `name` in memory, from a previous `set`
 - `=string` Literal string value, e.g. `"2"`, `=Fred`
- `%[** a remark]%`
 - Is a comment.

HTMX Blocks

- `%[*block, &blockname, end-re]%`
 - Reads following lines until a line matching regular expression `end-re` and puts the lines in variable `blockname`.
 - For `end-re`, use something like `^EOB`.
 - Variables in the block are not expanded at definition time, but instead when the block is expanded later.
 - Specifying the same `blockname` more than once *appends* content to the block definition.
 - Must be alone on a line. Blocks do not nest.
- Use blocks to put HTMX into a variable that can be...
 - expanded later, as in example 2.
 - expanded many times, e.g. for database iterators.
 - called as a macro with `%[*callv, blockname, ..]%`

Conditionals

- `%[*if,rel,varname,value,statement]%`
 - Expand `statement` if `varname` has relation `rel` to `value`
 - `rel` may be `gt lt eq ne ge le =~ !~ eqlc nelc`
 - `statement` may be a variable name or builtin, e.g. `*set, *if`, and may have arguments.
 - Comparisons are done as in Perl.
 - The `=~` and `!~` operators use regular expression match.
 - The `eqlc` and `nelc` operators ignore case.
- **Examples:**
 - `%[*if,=,moddate,="",*set,&moddate,date]%`
 - Sets `moddate` to the current date if it is blank.
 - `%[*if,eq,d,"0",*if,ne,sm,="",*fwrite,=%[m]%,sm]%`
 - If `d` is zero and `sm` is nonblank, write `sm` into file `m`.
 - `%[*if,gt,x0,=999,*subst,&w0,"^.*(..)$",=",$1"]%`
 - Drops all but the last 3 characters of `w0` and prefixes them with a comma.

More HTMX built-ins

- `%[*expand, varname]%`
 - Expand constructs in `varname`, output the result.
- `%[*expandv, &var, varname]%`
 - Expand constructs in `varname`, put result in `var`, output nothing.
- `%[*concat, &varname, value]%`
 - Concatenate `value` onto the value in `varname`, output nothing.
- `%[*ncopies, &varname, value, n]%`
 - Put `n` copies of `value` into `varname`, output nothing.
- `%[*subst, &varname, left, right]%`
 - Apply Perl substitution `s/left/right/ig` to the contents of `varname`, replacing its contents, output nothing.
 - `left` can be a regular expression.
 - `left` can contain parenthesized strings, used in `right` as `$1` `$2` etc.
 - Backslashes in the expression need to be doubled.
 - Slashes in `left` or `right` need to be prefixed by `\\`

HTMX Arithmetic

- `%[*increment, &varname, value]%`
 - Increment `varname`'s contents by contents of `value`, output nothing.
- `%[*decrement, &varname, value]%`
 - Decrement `varname`'s contents by contents of `value`, output nothing.
- `%[*product, &varname, value1, value2]%`
 - Multiply `value1` by `value2` and store in `varname`, output nothing.
- `%[*quotient, &varname, value1, value2]%`
 - Divide `value1` by `value2` and store integer in `varname`, output nothing.
- `%[*quotientrounded, &varname, value1, value2]%`
 - Divide `value1` by `value2`, round, store integer in `varname`, output nothing.
- `%[*scale, &varname, value1, value2, value3]%`
 - Store `int(.5+(value1*value3)/value2)` in `varname`, output nothing.

External files

- `[%fwrite,=file,varname]%`
 - Write contents of `varname` to `file`, output nothing.
- `[%fappend,=file,varname]%`
 - Append contents of `varname` to `file`, output nothing.
- `[%fread2,&varname,=file]%`
 - Read contents of `file` into `varname`, output nothing.
 - If input is not found, sets `varname` to empty string.
 - Does not expand values or blocks.

External Values

- `%[*shell,&x,abc]%`
 - Executes the shell command in variable `abc`, sets `x` to result, output nothing. If multiple lines are returned, change newline to the value of `_ssvsep` (default is space).
 - Example:
`%[*shell,&xdate,=filemodiso %[*inputfile]%)%`
might set `xdate` to "2016-07-04"
- `%[*urlfetch,&varname,=url]%`
 - Read contents of `url` into `varname`, output nothing.
 - (Think carefully about security if you do something like this.)

External Shell Scripts

- **Useful external shell commands**
 - Import values into expandfile
 - write your own as needed, in any language
 - filemodshort, filemodyear, filemodiso
 - filedaysold, filesizek
 - gifsize
 - firstletter, uppercase, lowercase
 - fmtnum
 - nargs

Miscellaneous

- `%[*format, &varname, fmtstring, val1, val2, ...]%`
 - Replace `$1 $2` etc in `fmtstring` with corresponding values.
 - Result in `varname`, output nothing.
- `%[*htmlescape, varname]%`
 - Output a HTML-escaped version of `varname`.
 - `<fred> => <fred>`
- `%[*warn, message]%`
 - Write a line to `STDERR`.
- `%[*dump]%`
 - Output the entire symbol table for debugging.

Special Files

- `config.html`
 - Defines values you use in many files.
 - Sequence of `%[*set,&var,=value]%` commands.
- **Included files and macros**
 - Examples: `htmlib.html`, `pagewrapper.html`
 - Standard formatting used in many pages
 - Write your own, copy others
 - Libraries

HTMX error messages

- `expandfile: x.htmx missing end of *block BLOCKNAME -- REXP`
- `expandfile: x.htmx need NUMBER]%`
- `expandfile: x.htmx unclosed quoted string beginning 'XXXX'`
- `expandfile: x.htmx missing CSV file 'FILE' ERR`
- `expandfile: x.htmx missing XML file 'FILE' ERR`
- `expandfile: x.htmx missing *include 'FILE' ERR`
- `expandfile: x.htmx missing *includeraw 'FILE' ERR`
- `expandfile: x.htmx cannot *fappend 'FILE' ERR`
- `expandfile: x.htmx cannot *fwrite 'FILE' ERR`
- `expandfile: x.htmx extra arguments X,Y,Z... to *BUILTIN`
- `expandfile: x.htmx invalid varname *set,&VARNAME`
- `expandfile: x.htmx unknown builtin *NAME,ARGS`
- `expandfile: x.htmx cannot open DBI:mysql:DB:HOST USER for query QUERY`
- `expandfile: x.htmx cannot prepare query QUERY ERRMSG`
- `expandfile: x.htmx cannot execute query QUERY ERRMSG`
- `expandfile: x.htmx cannot execute COMMANDLINE ERR`

- `expandfile: warning: x.htmx *set,&VARNAME varname is all digits, is = missing?`
- `expandfile: warning: x.htmx missing = before argument VARNAME`
- `expandfile: warning: x.htmx>w.html>imgtag 'imgtag_result' should begin with &`

Using HTMX Templates

I usually start with comments

```
[%]** created by Tom 02/28/15 **]%
```

Set initial values of some variables

```
[%*set,&title,="Daily report"]%
```

Define some blocks, which may refer to variables

```
[%*block,&content,^END]%
```

...

```
<h1>[%title]%/h1>
```

...

```
END
```

At the end, include a HTMX wrapper file that expands variables including blocks (that may expand variables).

```
[%*include,=pagewrapper.html]%
```

HTMX idioms

- Some useful code:
- Change a variable containing HTML so that `` tags at the beginning of the line are joined to the previous line.
 - `%[*subst, &menu, = "\n<\/li>", = "<\/li>"]%`
 - Escape slashes in `subst` args, since it uses `/` as delimiter.
 - The newline is input to the `subst` as backslash-n.
 - Input the backslash as double backslash.
- Use a value from the Makefile in a macro call
 - Say you want to define a relative path prefix `REL`
 - In the Makefile, use
`REL=../`
 - In the template, use
 `%[*callv, img2, = %[REL]%"icon.gif"]%`
 - Don't put `%[REL]%` inside the quotes, or it won't be expanded.

SQL Loops

- Extracting data from an SQL database:
 - `%[*sqlloop,&rs,tpt,="SELECT * FROM table1"]%`
- This statement
 - Performs the database query and expands `tpt` for each row after binding values to variable names in the symbol table like `table1.varname`
 - Computed values such as `COUNT` are bound to names like `.count`.
 - Database parameters come from the symbol table variables
 - `hostname`, `database`, `username`, `password`
 - These values are often set in `config.htmli`
 - Binds `_nrows` to the number of rows read.
 - Binds `_colnames` to a space separated list of the column names
 - Concatenates all expansion output into variable `rs`, outputs nothing.
- A warning is printed if no rows are selected, and execution continues.

Using *sqlloop


- **Some useful functions to use in templates:**
 - `%[*onchange,var,statement]%`
 - `%[*onnochange,var,statement]%`
- **Handy macro to dump what was bound, for debugging:**
 - `%[*callv,sqldump,string]%`

CSV loops

- A similar loop operates on each row in a CSV (comma separated values) file. (The file may be gzipped.)
 - `%[*csvloop,&resultvar,rowtpt,=filename]%`
 - First line of the file provides the column names
 - For each of the rest of the rows, parses items, binds to column names, expands `rowtpt` for the row, appends result to `resultvar`, outputs nothing.
 - Items in the CSV file are comma separated and may be quoted
 - example: `this,"is,an,example",12345`
 - three items:
 - `this`
 - `is,an,example`
 - `12345`
 - Binds `_nrows` to the number of lines read, after loop finishes.
 - Binds `_colnames` to space separated list of column names.

CSV Loop Example

```
%[*set,&rowno,=0]%  
%[*csvloop,&outvar,iter,=examp.csv]%  
%[*block,&iter,^END]%  
%[*increment,&rowno,=1]%  
row [%rowno]%;   [%col1]%   [%col2]%   [%col3]%   [%col4]%   [%col5]%;  
END
```



What I read from file "examp.csv" ← **text that is copied**
%[*includeraw,=examp.csv]% ← **builtin**

The column names are: %[_colnames]% ← **variable expansion**

We read in %[_nrows]% rows not counting the header

Formatted output:

%[outvar]% ← **expansion of variable set by CSV loop**

XML loops

- A similar loop operates on each item in an XML file. (File may be gzipped.)
 - `%[*xmlloop,&resultvar,tpt,=filename]%`
 - XML file has
 - outermost `<list> ... </list>`
 - containing a sequence of `<item> ... </item>`
 - Each item contains multiple fields
 - `<item>`
 - `<name>John Smith</name>`
 - `<addr>1234 Any Street</addr>`
 - `</item>`
 - For each item, binds fields to names like `item.name`, expands `tpt`, appends result to `resultvar`, outputs nothing.
 - Binds `_nxml` to the number of items read, after loop finishes.
 - Binds `_xmlfields` to space separated list of field names.
 - Instead of `<cite>` use `!!cite!!whatever!!/cite!!`
 - Instead of `á` use `&aacute;`

XML loops cont.

- **An alternate form of the loop allows an XPath**
 - `%[*xmlloop, &resultvar, tpt, =filename, xpath]%`
 - **for cases where the XML structure is more complex.**
 - **if xpath is not specified, the default is `" /* /* "`**
 - **for each item found by the XPath arg,**
 - **bind the values of sub-items `" ./ * "`**
 - **bind the values of attributes of items `" ./ @ * "`**
 - **example file, access items with `"*/computers/computer"`**

```
<?xml version="1.0" encoding="UTF-8"?>
<computer_group>
<name>All Managed Clients</name>
<computers>
<size>119</size>
<computer>
<name>QA2 MacBook</name>
<serial_number>54321</serial_number>
</computer>
<computer>
<name>Rocky</name>
<serial_number>12345</serial_number>
</computer>
...
```

File System Loops

- A similar loop iterates over a directory.
- `%[*dirloop,&outvar,iter,dirpath,starrex]%`
- Lists directory `dirpath` and expands block `iter` for each file matching regular expression `starrex`; output is appended to variable `outvar`, outputs nothing.
- e.g. `%[*dirloop,&out,it2,="/home/jack/xx",="*.html"]%`
- **Binds variables to the values of `status()` info on each file:**
 - `file_name, file_type, file_dev, file_ino, file_mode, file_nlink, file_uid, file_gid, file_rdev, file_size, file_atime, file_mtime, file_ctime, file_blksize, file_blocks, file_sec, file_min, file_hour, file_mday, file_mon, file_year, file_wday, file_yday, file_isdst, file_datemod, file_modshort, file_sizek, file_age`
- **Variable `file_type` is set to 'f' for file, 'd' for dir, 'l' for link**

SSV Loops over variables

- **A similar loop iterates over a list of items in a string value.**
("ssv" stands for "space sparated variables")
- `%[*ssvloop, &outvar, iterblock, ssv]%`
 - **Breaks `ssv` into tokens and expands `iterblock` for each, binding `_ssvitem` to the token (skips null tokens).**
 - **Does not modify `ssv`.**
 - **Binds `_nssv` to the number of tokens processed.**
 - **Token separator is the value of `_ssvsep`, default is space.**
 - **Result is stored in `outvar`.**
 - **Outputs nothing.**
- `%[*popssv, &var, &ssv]%`
 - **Takes first value from `ssv`, puts it in `var`, rewrites `ssv` to remove value, outputs nothing.**

External Programs

- Invoke any shell program or script and capture its output.
- **Example:**
 - `%[*shell,&result,=filemodshort myfile.htmx]%`
- **Called from Perl, output has NL changed to space**
- **Useful programs (can be written in any language):**
 - `filemodshort file => 03/27/07`
 - `filesizek file => 33`
 - `nargs a bb c dd e f g => 7`
 - `gifsize pic.jpg => "pic.jpg" width="75" height="75"`
 - `lowercase ABcdE => abcde`
 - `nrandomlines x.htmx n (n lines from x.htmx separated by |)`
 - `firstletter applecart => a`
 - `filedaysold filename => 12`
 - `fmtnum 123456 => 123,456`
 - `grep, sed, date, cut, awk => (Unix command output)`
- **Write your own as needed**

Macro Example

Call: `%[*callv,imgtag,="abc.jpg",="AAA",="Bbb"]%`

Generates: ``

```
%[** macro for generating an image tag **]%  
%[*block,&imgtag,^END]%
```

define block "imgtag"

```
%[** parameters: ]%  
%[** param1 - file path, same rel path must work in source and obj]%  
%[** param2 - alt tag]%  
%[** param3 - title tag]%  
%[** output: none]%  
%[** sets: imgtag_result (resulting image tag)]%  
%[** ----- **]%  
%[*shell,&y,=gifsize %[param1]]%
```

**external shell call to "gifsize"
(a program in my library)**

```
%[*if,eq,y,="",*concat,&y,param1]%  
%[*set,&imgtag_result,=""]%  
END
```

Multics Formatting

- Enabled if the `-mult` switch is specified to `expandfile`
- Optional constructs for
 - `pathname: {=text=}`
 - expands to `text`
 - `command: {:text:}`
 - expands to `text`
 - `code: {+text+}`
 - expands to `text`

Multics Links

- Optional constructs (with `-mult` switch) for
 - External link reference `{!tag anchor text!}`
 - See `loadext.sql` for external tags
 - Internal link reference `{@file anchor text@}`
 - See `loadpages.sql` for internal files
 - Link to Glossary `{{tag anchor text}}`
 - See `g1.sql` for glossary tags
 - Link to `multicians.html` `{[nametag Name]}`
 - See `loadm.sql` for Multicians tags
- Each construct expands to a hyperlink with an HTML `title` attribute from the database. (Database parameters from the symbol table.)

Example from Multics Website

- **Example story file:** `manning.htm`
 - Sets some variables, e.g. `title`
 - Defines two blocks, `body` and `extratail`
 - Inserts `pagewrapper.html`
- **Source features**
 - Paragraphs have `<p>` and `</p>` (see *HTML Tidy*)
 - Uses `Corbató`; `and` `&`; in text
- **Multics features activated by `-mult` switch**
 - Author para has `{[Manning Eric Manning]}` and has semantic tag `class="author"`
 - Special link `{@-manning.html Next Story@}`

expandfile

- **what:** Perl program in your `~/bin`
- **how to invoke:**
`expandfile -config x.html x.htmlt > x.html`
- **what it does:** expands a template
- **who calls it:** `Makefile`, shell scripts
- **how it works**
 - *get args, read config file*
 - *read template file*
 - *scan for blocks, register them, remove from template*
 - `&expandstring()` *the rest*
 - *write the result on `stdout`*

Other uses for `expandfile`

- `expandfile` doesn't know or care about the format of its input and output: *it does not know about HTML*.
- Can use it to create any kind of text file. Examples:
 - input to `GraphViz` and `rrdtool`
 - `procmail` control file
 - `mysql` input
 - XML data: Google site map, RSS feed
 - shell scripts
- Super Webtrax (SWT)
 - Web site traffic analysis report
 - Loads web server log into SQL, expands templates that query the database and write HTML reports

&expandstring()

- **what:** Perl function provided in `thvve.pm`
- **how to invoke it:**

```
$result = &expandstring($tpt, \%symtb);
```
- **what it does:** returns expansion of template
- **who calls it:**
 - *expandfile, special expanders, various CGIs*
- **how it works**
 - *Parses and interprets HTMX language*
 - *Uses and sets symbol table*
 - *Returns string as output*
- **run auxiliary function for block expansion first:**

```
$newtpt = &expandblocks($tpt, \%symtb);
```

Using `&expandstring()`

- **Perl skeleton**

```
use thvve.pm;
my %symtb;
# set up $symtb{xxx} with contents, computed somehow
# read in a template into $tptstr
$result = &expandstring($tptstr, \%symtb);
print "$result\n";
```

- **The Perl program has no output formatting built into it.**
- **The template does all formatting; could use HTML, CSV, or some other language. (Same data could be formatted in more than one language.)**
- **The coupling between the Perl program and the template is the list of hash keys / variable names in `%symtb`.**

Uses for `&expandstring()`

- **Online CGI programs in Perl call `&expandstring()` to**
 - **Generate dynamic web pages**
 - *mail form*
 - *registration*
 - *statistics*
 - *RSS feed formatting*
 - *management*
 - *result and error pages*
 - **Format mail messages**
 - *mail sending*
- **Offline Perl programs call `&expandstring()` to expand templates.**

HTMX / PHP similarities

- **Both extend HTML**
- **Both have variables and functions**
- **Variable values are strings in each**
- **Both have user-defined functions and libraries**
- **Both can include files**
- **Both have MySQL integration**

HTMX / PHP differences

HTMX

- **Static compilation**
- **Parse once**
- **Errors found at compile**
- **35 builtin functions**
- **Web server agnostic**

- **Blocks**
- **Wrapper templates**
- **Loops on SQL, CSV, XML, File System, and SSV**

PHP

- **Runtime interpreter**
- **Parse on every view**
- **Errors found at runtime**
- **>5000 builtin functions**
- **Web server integrated**
- **Arg sanitization risk**
- **Integrated with FORM**

Design Choices for HTMX

- **Static pages**
- **Uses existing facilities**
 - **Can be used to extend HTML**
 - **Doesn't know HTML syntax**
 - **Uses the file system**
 - **Uses UNIX tools**
 - **extend language via shell commands**