Caucus Markup Language

Version 3.1
Reference Guide

Screen Porch LLC.

4031 University Drive, Suite 200
Fairfax, VA 22030-3409
http://screenporch.com

Revision Date: 18 February 1997

1. INTRODUCTION

This document is the reference guide for CML, the **C**aucus **M**arkup **L**anguage. CML is a "mark up" language that combines HTML tags with simple programming constructs and database functions. The CML language interpreter is the core of the World Wide Web interface to the Caucus conferencing system (hence the name).

This guide assumes considerable familiarity with HTML, the World Wide Web, Web browsers, and the Caucus conferencing system. For more information about Caucus, see the Screen Porch home page at http://screenporch.com.

This is a work-in-progress, and will be frequently revised. This edition corresponds to the CML interpreter provided with the "Caucus 3.1.1" package. The author may be contacted at roth@screenporch.com.

1.1 What's New in version 3.1

Since this document is being revised constantly, this section briefly describes new features added to CML, indexed by date.

8 August 1996

$replace()	4.8	replace characters in string
$readfile()	4.4	read contents of any file
$inc()	4.2	"include" directive arguments
$if()	4.6	triadic operator
$t2amp()	4.17	escape &'s in HTML text
$form()	4.2	extended to support multi-part forms
$peo_names()	4.13	find people by name
$per_lastin()	4.11	time last in a conf
$it_howmuch()	4.14	how much of an item seen by a user?
$lower()	4.8	convert to lower case
$upper1()	4.8	upper case 1st letter of each word

1 November 1996

$my_text()	4.12	when is my text considered new?
$set_my_text()	4.12	set the above
$ad_author()	4.16	prepare a psuedonymous author name
$t2mail()	4.17	filter e-mail into "mailto:" address
$item_var()	4.18	value of item variables
$set_item_var()	4.18	set value of item variables
$per_real()	4.11	"real name" of a userid
$cml_dir()	4.3	CML directory of current URL

$open()	4.4	open a file
$readln()	4.4	read a line from an open file
$writeln()	4.4	write text to an open file
$close()	4.4	close an open file
$output()	4.4	redirect HTML output
$copy2lib()	4.4	copy file to file library
$safehtml()	4.17	"reduced HTML" filter
$find_it()	4.19	search list of items
$find_filter()	4.19	display "hits" from searches
$page_save()	4.20	save info to regenerate this page
$page_get()	4.20	get saved page information
$page_return()	4.20	prepare to return to saved page
$page_caller()	4.20	who "called" a saved page?
elif directive	5.4	extend if and else directives

14 February 1997

$wrap2html()	4.17	better handling of “wordwrapped” text
$str_index()	4.8	string searching
$str_revdex()	4.8	reverse string searching
$str_sub()	4.8	substring manipulation
$escquote()	4.17	escape double-quote marks
$wraptext()	4.17	word wraps arbitrary text
$less()	4.7	“less than” comparison

FB00024393

2. PURPOSE OF CML

The Caucus conferencing system was first released in 1986 as a text-based, command driven conferencing (groupware) package. Over the next 8 years, Caucus versions 1 and 2 were extended in a variety of ways that made it extremely customizable -- but still fundamentally text-based.

With the enormous growth of the World Wide Web in 1994-95, it became clear that a Web-based interface for Caucus could greatly increase its ease of use, and its popularity. At the same time, the Web lacked any significant discussion or conferencing tools, and it was clear that a Web interface for Caucus could fill this gap.

Version 3.0 of Caucus was developed from the ground up as web-based client-server system. It was designed to serve Caucus conference information to an HTTPD server, which in turn feeds HTML to any Web browser. But the Caucus server (called "swebd") needs to know what data to serve, and how (what format) to serve it in. This is the purpose of CML.

CML pages (files) are analogous to HTML pages. They contain Caucus directives (e.g., "display the text of such-and-such response) in an HTML-like format. They may also contain embedded HTML.

When a Web user wants to access (or add to) a Caucus conference, s/he points the browser at a special "entry" HTML page. This entry page points to a CML page. (The actual implementation of "pointing to a CML page" is done via the Web CGI standard.) CML pages point to other CML pages, exactly analogous to the way HTML pages point to other HTML pages.

When an HTTPD server gets a request for a CML page, it passes the request on (via CGI) to the Caucus swebd server. Caucus interprets the contents of the CML page, *producing a dynamic HTML page*, and passes it in turn on to the HTTPD server, which sends it to the browser.

3. WHAT'S IN A CML PAGE

Each CML page (or file) describes a page that will appear on the user's Web browser. (In some cases it just produces an HTTP "Location" directive which points in turn to another CML or HTML file.) CML can be thought of as a superset of HTML. More precisely, HTML is embedded in CML scripts; swebd does not actually understand or parse the HTML codes. A CML page contain 4 kinds of text:

1. Comments. In the Unix tradition, all lines beginning with "#" are comments and are ignored. Entirely blank lines are also ignored.

2. HTML code. All lines beginning with a double quote (") are parsed for CML functions, but are otherwise passed on to the browser unchanged. (The quote is removed.) There may be leading blanks before the quote; they are ignored.

3. CML functions. Strings of the form $xyz(), $xyz(value), or $(value) are parsed by swebd, and replaced by the appropriate Caucus values.

4. CML directives. Directives are like C program code: they describe actions to be taken. Directives start with one of the keywords "if", "elif", "else", "for", "count", "while", "set", "include", "return", "break", or "end".

A single logical line in a CML file may be broken across several physical lines; a "\" as the last character means "continued on next (physical) line". Most of the time this is not needed, since HTML mostly ignores line boundaries. However, the "\" is useful for assembling long lines that will appear inside HTML <PRE> code, or to improve readability of the CML code.

Here's a sample CML page, typical of a page a Web Caucus user would see early on:

```
#
#---CENTER.CML.   "Caucus Center" Page.
#
#   Overview of (and initial entry to) conferences.
#
#------------------------------------------------------------------------

if $empty ($(href))
   include $cml_dir()/startup.i center.cml
end

set nch $unique()
set nxt $page_save (1 center.cml \
              $arg(2)+$arg(3)+$arg(4)+$arg(5)+$arg(6)+$arg(7)+$arg(8) \
              # $(center_name) )
set last_conf x

#---HTML declaration, header, and BODY tag.
"Content-type: text/html
"
"<HTML>
"<HEAD>
"<TITLE>$(center_name)</TITLE>
```

FB00024395

```
"</HEAD>

"<BODY $(body_bg) >

#---Caucus header.
include $(dir)/header.i

#---Tell the user what this page is about.
"<P>
"<TABLE WIDTH=100% CELLSPACING=0 CELLPADDING=0>
"<TR>
"<TD><FONT SIZE=+1><B>Caucus Center</B></FONT></TD>
"<TD ALIGN=right>
   include $(dir)/youare.i
"</TD>
"</TABLE>

"<P>
"From here, you may go to specific conferences, or
"<A HREF="$(href)/allconfs.cml?$(nch)+$(nxt)+x+x+x+x+x+x">
"see a list of <B>all</B> conferences</A> on this host.
"<P>

#---Prepare to actually put up various kinds of links to the
#   conferences.  Create some variables with lists of
#   conference names.  Apply $cl_list() to the entire list
#   of conferences.
#     L_CONFS are the "popular" conferences.
#     M_CONFS are from the user's personal conference list
set l_confs $file($(inc)/l_confs.i)
set m_confs $user_var($userid() my_confs)
set ignore  $cl_list ( $(l_confs) $(m_confs) )

#---The various ways of getting to the conferences all appear
#   as numbered entries, within one large table.
#   To avoid unpleasant spacing, and because the "JOIN" choice requires
#   being in a <FORM>, the entire table must be inside a <FORM>.
"<FORM METHOD=POST ACTION="$(href)/centerf.cml?$(nch)+$(nxt)" NAME="joiner">

"<TABLE CELLSPACING=0 CELLPADDING=0 >

#---Personal conference list access:
set way_in 1
include $(dir)/cen_pers.i $(way_in)


#---"Popular" conference access:
if $sizeof ($(l_confs))
   include $(dir)/cen_pop.i way_in
end


#---Type a conference name directly:
set way_in $plus ($(way_in) 1)
include $(dir)/cen_type.i $(way_in)


```

```
#---See a list of all conferences:
#set way_in $plus ($(way_in) 1)
#include $(dir)/cen_all.i $(way_in)

"</TABLE>
"</FORM>
"<P>

#---Advertisement:
include $(dir)/cen_adv.i

"</BODY>
"</HTML>
```

4. CML FUNCTIONS

CML contains a large number of functions. These functions have several purposes:

1. Extract data (from the Caucus conference database) for display.
2. Manipulate or compare data (such as addition, subtraction, testing equality, etc.)
3. Put new data back into the Caucus database.
4. Maintain "state" information between CML pages.

All of the CML functions are described below. They have been broken up into related categories, for ease of reference. There is an alphabetical index of functions at the end.

The syntax of a function must always be "$name(arguments)". There must be no spaces between "$" and "name". Spaces may be used freely around the "(" and ")". Anything (including spaces or other functions) may be in the *arguments*. Some functions have no arguments.

If you wish to display a "$" in your HTML text, and not have it be confused with a CML function, escape it with a preceeding "\", i.e. "\$".

4.1 CML variables

The simplest CML function is "evaluate this variable". It looks like this:

```
$(name)
```

This means "evaluate the variable *name*, and place its value here". For more information about variables, see the CML directives "for", "count", and "set" in section 5.

4.2 CML state functions

The CML state functions are the glue that ties a group of CML pages and a Sweb server together. To understand more about why they exist, see the design document "The Web Caucus". For the CML author, it is only necessary to understand **where** they must be used.

$host()

Evaluates to the host name (and http port number) of the current host. This is a useful way to build HTML links that require the current host name, and still keep your CML code portable. Example:

```
"<A HREF="http://$host()/dir/page.html">some text</A>
```

$pid()

Evaluates to the pid (process id) and security code for the swebd server that is dedicated to your browser. You **must** include this in links to CML pages. Example:

```
"<A HREF="http://$host()/sweb/swebsock/$pid()/SP/test.cml?15+bye">
"     name</A>
```

$arg(n)

Evaluates to the *N*'th argument to this CML page. In the previous example, clicking on link "name" will bring up the CML page test.cml. In test.cml, $arg(1) will then evaluate to "15", and $arg(2) will evaluate to "bye".

$inc(n)

Evaluates to the *N*'th argument to this "include" file (see section 5).

$form(name)

When a CML page is the "recipient" of an HTML form (as in <FORM ACTION="...">), the form data is available through the CML $form function. The function evaluates to the data entered by the user in field *name* (as in NAME="*name*" in an <INPUT> or <TEXTAREA> HTML tag), or (in the case of TYPE="submit" fields) to the VALUE string for the button with NAME=*name*. If there are multiple values for the field *name* (as in a <SELECT MULTIPLE> field), the values are concatenated together, separated by single spaces.

The $form() function transparently handles both standard ("application/x-www-form-urlencoded") and "multipart/form-data" forms. $form() may be used only with METHOD= POST forms.

Multipart forms may be used with some browsers to upload an entire file, with an HTML tag of the form <INPUT TYPE="file" NAME="*name*">. In this case, $form(*name*) evaluates to the name of a temporary file on the server host. (The uploaded data has been placed in that file). The temporary file will be automatically deleted when the swebd process exits (i.e., when the user's session is over). The original name of the file is also available as $form(*name*.upload)

$debug(n)

N = 1 turns on debugging, which writes data to a logging file in /tmp. *N* = 0 turns off debugging. The default is 0.

$caucus_id()

Evaluates to the name of the caucus userid, i.e. the userid that owns the Caucus files.

4.3 Brower and server information and control

$userid()
: Userid of the current user.

$cml_dir()
: Evaluates to the directory name of the current CML file. For example, if the URL is
http://screenporch.com/spi/swebsock/0008404/0083664/SP31/center.cml?1+x+x
then $cml_dir() will evaluate to "SP31".

$http_user_agent()
: Contents of the CGI environment variable HTTP_USER_AGENT. Usually a multi-word string that describes the browser client program.

$goodbye()
: Tells the swebd server dedicated to this user to change its timeout period to one minute. This is a graceful way to exit Caucus, and lowers system load. It is not required, the swebd server will eventually timeout by itself.

$unique()
: Return a unique number each time. Useful for tagging distinct instances of a particular page.

$version()
: Returns version number of Caucus server software (e.g., "3.1.04").

$is_passwd()
: Evaluates to '1' if a password changer program was defined in the configuration file swebd.conf, and '0' otherwise.

$reval(string)
: Recursively evaluates *string* for CML functions. If *string* contains a CML function, which when evaluated expands to a CML function, reval() makes sure that *string* is continually interpreted until no CML functions remain.

Without reval(), CML text is scanned only once for CML functions.

$protect(string)
: Prevents certain CML functions from taking effect. Any CML functions in *string* operate in a "protected" mode. This is useful, for example, in evaluating CML code that may have been placed (by a user) in the text of an HTML response.

Functions which have no effect when evaluated inside $protect() include: shell(), silent(), passwd(), set_wrap(), any set_co...(), any set_it...(), any set_my...(), any ad_...(), any chg_...(), set_user_var(), and set_conf_var().

4.4 File Access

$file(name)

Include the entire text of file *name* at this point.

The $file() function should only be used to include relatively short (a couple of lines, maximum) files, such as when you need to include the contents of a file in the middle of an HTML or SET string that you are building. *Name* is relative to the CML_Path directory specified in the swebd.conf file. (See the Caucus installation guide for details.)

If you need to include a large file, or one that contains CML directives, see the "include" directive in section 5.

$readfile(name)

Evaluate to the entire contents of text file *name*. *Name* should be the full pathname of a file on the server host. Whereas $file() is meant as a way to include additional CML code in a page, $readfile() is meant for reading data that will somehow be processed or displayed by a CML page.

$write(name text)

Write *text* to file with absolute pathname *name*. Overwrites previous contents of *name*, if any.

$append(name text)

Append *text* to file with absolute pathname *name*.

$dosfile(name)

Truncates *name* to the first 8 characters, and replaces all dots (".") with underscores ("_"). Useful when *name* refers to a file on the client machine.

$copy2lib(file libname)

Copies *file* (a full pathname) to a new file called *libname*, in the Caucus file library. (See the parameters Caucus_Lib and HTTP_Lib in the swebd configuration file swebd.conf for more information about the Caucus file library.) *Libname* may contain sub-directory names, and is always treated as relative to the root of the Caucus file library. Sub-directories are created automatically. Thus a *libname* of "demo/xyz" would copy *file* to a file called "xyz" in a sub-directory "demo" under the Caucus file library, and would automatically create the "demo" directory if needed.

The function evaluates to the full URL of the newly created file, thus making it possible to make the file immediately available on the Web in any subsequently produced HTML.

$open(name perm)

Open a file *name* for reading (if *perm* is "r"), for writing (if *perm* is "w"), or to append to (if *perm* is "a"). Evaluates to a number which is the file "handle", or to "0" if the file could not be opened.

$readln(handle var)

Read a line from the file open on *handle*, and put the text into variable *var*. Evalutes to "1" if successful, or to "0" on end-of-file.

$writeln(handle text)

Writes *text* to the file open on *handle*. Evaluates to "1" on success, or "0" if *handle* does not refer to an open file.

$close(handle)

Close file open on *handle*.

$output(name mask)

Normally, CML lines that begin with a double-quote (") are interpreted and sent directly to the user's browser. The $output() function redirects this text, and writes it to a file *name*, instead. *Mask* is the numeric Unix file permission mask, e.g. a value of "644" means read/write owner, read group, and read world.

The redirection takes effect on all quoted lines that follow the use of $output(). Another call to $output(), with no arguments, returns subsequent output from quoted lines to the browser, in effect "closing" the file.

4.5 Shell access

CML provides two functions for accessing the Unix shell (and thereby running commands or scripts from the shell). Both functions run a command or script with the effective userid of the httpd server.

$shell(command)
Runs *command* in a shell. The function evaluates to the output from *command.* Example:

```
"It is now: $shell(date)
```

$silent(command)
Runs *command* in a shell. The output is ignored. The function evaluates to nothing, i.e. it effectively disappears. The example logs a user's userid to a temporary file.

```
" $silent(echo $userid() >>/tmp/log)
```

4.6 Comparisons and logical functions

$and(a b ...)
Evaluates to the logical "and" of *a* and *b* and ... May have any number of arguments.

$or(a b ...)
Evaluates to the logical "or" of *a* and *b* and ... May have any number of arguments.

$not(a)
Evaluates to the logical negation of *a*.

$equal(x y)
If *x* and *y* are identical (they may be numbers or strings), evaluates to "1". Otherwise it is "0".

$not_equal(x y)
Reverse of $equal().

$empty(str)
Evaluates to "1" if *str* is completely empty, and "0" otherwise.

$not_empty(str)
Evalutates to "1" if *str* is not empty, and "0" if it is completely empty.

$if(a b c)
If *a* is true, evaluates to *b*. Otherwise, evaluates to *c*. The classic triadic "if then else" operator.

4.7 Mathematics

$plus(a b)
Evaluates to the sum of numbers *a* and *b*.

$plusmod(a b x)
Evaluates to sum of *a* and *b*, modulo *x*.

$minus(a b)
Evaluates to the difference, *a* - *b*.

$mult(a b)
Evaluates to the product of *a* and *b*.

$divide(a b)
Evaluates to the integer quotient of *a* / *b*.

$greater(a b)
Evaluates to "1" if *a* is greater than *b*. Otherwise "0".

$gt_equal(a b)
Evaluates to "1" if *a* is greater than or equal to *b*. Otherwise "0".

$less(a b)
Evaluates to "1" if *a* is less than *b*. Otherwise "0".

$between(a x b)
Evaluates to "1" if *x* is between *a* and *b* (*a* <= *x* <= *b*). Otherwise "0". Very useful for processing the result of server-side image maps.

$max(a b)
Evalutes to the larger of numbers *a* and *b*.

$min(a b)
Evaluates to the smaller of numbers *a* and *b*.

4.8 String Manipulation

$upper(words)
Converts all the text in *words* to upper case.

$upper1(words)
Converts the first letter of each word in *words* to upper case.

$lower(words)
Converts all the text in *words* to lower case.

$newline()

Evaluates to a newline character. Useful inside arguments to functions such as $t2hbr(), $ad_item(), etc.

$word(n str)

Evaluates to word number *n* of string *str*. Words are separated by one or more spaces. The first word is word number 1.

$rest(n str)

Evaluates to the "rest" of the words in a string, i.e. word number *n* through the end of *str*, inclusive.

$tablefind(word str)

Look for *word* in *str*. If it is identical to a single word, evaluate to the number of that word in *str*. Otherwise '0'.

$sizeof(str)

Evaluates to the number of words in string *str*.

$width(str)

Evaluates to the width (equivalent number of single-width characters) of *str*. Double-wide kanji have a width of 2.

$pad(num)

Evaluates to *num* blanks. Generally only useful inside <PRE> text.

$replace(a b c)

Each of the strings *a* and *b* must either be single characters, or else the base-ten numeric representation of a single character. $replace() evaluates to string *c*, but with each instance of character *a* replaced by character *b*.

$str_index(what text)

Find the first occurrence of the (one-word) string *what* in string *text*. Evaluates to position number of *what* in *text*. (The first position is 0.) Evaluates to “-1” if not found.

$str_revdex(what text))

Find the **last** occurrence of the (one-word) string *what* in *text*. Evaluates to position number of *what* in *text*. (The first position is 0.) Evaluates to “-1” if not found.

$str_sub(pos len text)

Evaluates to a substring of *text*, starting at position *pos*, *len* characters long.

4.9 Conference List Information

There is a family of functions that provide basic information about conferences. All of these conferences begin with "$cl_", to indicate that they refer to information about a list of conferences ("cl", as in **c**onference **l**ist).

$cl_list(names)

Evaluates to a list of conference numbers. If *names* is empty (i.e., nothing), $cl_list() evaluates to the list of all conferences on the host. If *names* contains one or more words, $cl_list() evaluates to the list of conferences that match any of the words in *names*. Example:

```
for cnum in $cl_list(web x)
```

$cl_list() becomes the list of all conferences whose names start with "web" or with "x". (The "for" loop thus sets `cnum` to each such conference number in turn.)

Note: the list of conference numbers is sorted, not by number, but by the name of each conference, regardless of the order of the arguments to $cl_list().

$cl_num(name)

Evaluates to the number of the conference whose name matches *name*. (An abbreviation is a match). *Name* must have been in the list of conferences generated by the most recent use of $cl_list().

$cl_name(num)

Evaluates to the name of conference number *num*. The name will always be in lower-case. $cl_list() must be called before $cl_name() can be used.

$cl_access(num)

Evaluates to the user's access level to conference *num*. 0 means the user is excluded from the conference, 1 means read-only access, 2 means full "include" access, and 3 means organizer access.

4.10 Conference Organizer Information

Another family of functions relates to information about a conference that gets set (or changed) by the **c**onference **o**rganizer. In all of these functions, *num* is the conference number.

$co_org(num)
Evaluates to the userid of the primary organizer of the conference.

$co_greet(num)
Evaluates to the text of the "greeting" for the conference. In the original (text) Caucus interface, the greeting was displayed every time a person entered a conference.

$co_intro(num)
Evaluates to the text of the "introduction" for the conference. In the original Caucus, the introduction was displayed when a person tried to join a conference for the very first time. The introduction offers more information about the conference, to help a person decide if they really wish to join the conference.

$co_add(num)
Evaluates to "1" if ordinary users can add an item, or "0" otherwise.

$set_co_add(num add)
If *add* is non-zero, ordinary users may add new items. If *add* is "0", they may not.

$co_change(num)
Evalutes to "1" if ordinary users can change their own responses. Otherwise "0".

$set_co_change(num chg)
If *chg* is non-zero, ordinary users may change their responses. If *chg* is "0" they may not.

$co_visible(num)
Evaluates to "1" if conference name is visible to non-members in conference lists. Otherwise "0".

$set_co_visible(num vis)
If *vis* is non-zero, the conference name is visible. If *vis* is "0", the conference is invisible to non-members.

$co_userlist(num)
Evaluates to the text of the conference "userlist".

$set_co_userlist(num list)
Set the text of the conference "userlist" to *list*.

4.11 Information about a Person

Another family of functions provides information about a particular person who is registered with Caucus. All of these functions begin with "$per_".

$per_name(id)
Evaluates to the full name of the person with userid *id*.

$per_intro(id)
Evaluates to the text of the "brief introduction" of userid *id*.

$per_phone(id)
Evaluates to the telephone number of userid *id*.

$per_laston(id)
Evaluates to the date and time that userid *id* was last on (last using) Caucus.

$per_lastin(id cnum)
Evaluates to the date and time that userid *id* was last in conference *cnum*.

$per_real(id)
Evaluates to the "real name" (as registered in the server host system password file) of user *id*. If there is no "real name", it evaluates to the empty string. (Note: this function is only meaningful when the Web userids are derived from the system password file userids. See the Caucus Installation Guide for more information.)

4.12 "My" Information

The "$my_" and "$set_my_" functions relate to registration information about the current user.

$my_exist()
Evaluates to "1" if the current user is registered with Caucus, and "0" otherwise.

$my_name()
Evaluates to the current user's full name.

$my_phone()
Evaluates to the current user's telephone number.

$my_intro()
Evaluates to the text of the current user's "brief introduction".

$my_laston()
Evaluates to the time and date the current user was "last on" Caucus.

$my_text()
Evaluates to a number which represents when a person's items or responses should appear as "new". 0 means "later" (only after someone else adds a response), 1 means now (immediately becomes new), and 2 means never (it is immediately treated as "seen").

$set_my_text(n)
Sets the value of my_text, as defined above, to *n*.

$set_my_name(name)
Sets the current user's full name to *name*. (Will not change the user's name if *name* is empty.) Evaluates to '1' on success, '0' on failure. (Fails if attempting to create a new user, and the maximum total number of users for this license has been reached.)

$set_my_phone(number)
Sets the current user's telephone number to *number*. (May be set to nothing). Evaluates to nothing.

$set_my_intro(text)
Sets the current user's brief introduction to *text*. (May be set to nothing). *Text* may contain newlines. Evaluates to nothing.

$passwd(id newpw oldpw)
Change the password for user *id*, to *newpw* (from *oldpw*). Evaluates to a success code: 0 means success; 1 means a missing argument; 2 means *oldpw* is wrong; 7 means the password changer was not enabled for this web site; 8 means a system configuration error.

$passcheck(id pw)
Evaluates to '1' if *id* and *pw* are a valid password pair, and '0' otherwise. If $passcheck() fails, most CML functions that reference actual Caucus data are disabled. (If $passcheck() is called again and succeeds, the functions are enabled. Functions are enabled by default.)

4.13 Information about groups of people

$peo_members(cnum)
Evaluates to a list of userids that are members of conference *cnum*. The userids are sorted by "last name" of the actual users.

$peo_names(cnum names)
Evaluates to a list of userids of people who match *names*. A person matches if every word in *names* is an initial substring of some part of their name. If *cnum* is non-zero, matching people must also be a member of conference *cnum*.

4.14 Item Information

The "it_" and "set_it_" functions provide or manipulate information about an item, or items, in a conference, or the user's participation in a conference. *Cnum* always refers to the conference number. *Inum* is a particular item number. *Rnum* is a particular response number.

$it_member(cnum)
Evaluates to "1" if the current user is a member of the conference.

$it_join(cnum)
Make the current user a member of the conference. Evaluates to "1" if joining is successful, and "0" otherwise.

$it_resign(cnum)
Resign (remove) the user from the conference. Evaluates to nothing.

$it_last(cnum)
Evaluates to the number of the last item in a conference, i.e., the number of items.

$it_icount(cnum)
Evaluates to the actual number of (non-deleted) items in a conference.

$it_inew(cnum)
Evaluates to the number of new (and undeleted) items in a conference.

$it_rnew(cnum)
Evaluates to the **total** number of new responses in a conference.

$it_iforgot(cnum)

Evaluates to the number of forgotten items in a conference.

$it_wnew(cnum)

Evaluates to the number of items that have 1 or more new responses.

$it_iunseen(cnum)

Evaluates to the number of unseen items.

$it_listinew(cnum)

Evaluates to a space-separated list of the new items in a conference. This list appears in "triplet" form. This means that each item is represented by three numbers: a conference number, an item number, and the number of the first relevant response. For example, if conference 17 has two new items, 5 and 6, $it_listinew() would produce the string "17 5 0 17 6 0". To parse triplet lists, use the functions $word() and $rest().

$it_listrnew(cnum)

Evaluates to a "triplet" list of the new responses in a conference. The response number in a triplet is the first new response in the relevant item.

$it_listiunseen(cnum)

Evaluates to a "triplet" list of the unseen items in a conference. The response number is always 0.

$it_exists(cnum inum)

Evaluates to "1" if the item exists, and "0" otherwise.

$it_visib(cnum inum)

Evaluates to "1" if the item is visible to the user, i.e. has not been deleted or "forgotten". Otherwise "0".

$it_new(cnum inum)

Evaluates to "1" if the item is new to this user, i.e. it has a higher number than the highest item the user has seen. Otherwise "0".

$it_unseen(cnum inum)

Evalutates to "1" if this item is not new but has not been seen by the user. Otherwise "0".

$it_resps(cnum inum)

Evaluates to the number of responses. If the item does not exist (or was deleted), evaluates to -1. An item without any responses evaluates to "0".

$it_newr(cnum inum)

Evaluates to the number of the first response on this item that is *new* to this user. If no responses are new, evaluates to the number of responses + 1.

$set_it_seen(cnum inum rnum)

Marks all responses through *rnum* as "seen" by this user. To mark an item as "unseen", use an *rnum* of -1. To mark an item as "forgotten", use an *rnum* of -2.

$it_frozen(cnum inum)

Evaluates to "1" if the item is frozen, and "0" otherwise.

$set_it_frozen(cnum inum value)

A *value* of 1 freezes the item. A *value* of 0 "thaws" it.

$it_howmuch(cnum inum userid)

Evaluates to the number of responses seen by user *userid* to item *inum* in conference *cnum*. A value of -1 means the item is new to that user; -2 means the user has forgotten that item.

Note: to ease the writing (and reading) of CML pages, all of the $it_ functions that take two arguments (such as it_visib(), it_resps(), and it_newr()) may be written with *no* arguments. This means "re-use the exact same arguments as in a previous instance of one of these functions". **Warning**: results may be unpredictable if other $it_...() functions (those with more than two arguments) are called in between.

4.15 Response Information

The "re_" functions provide information about a particular response. As in the previous section, *cnum* refers to a conference number, *inum* to an item number, and *rnum* to a response number.

$re_exists(cnum inum rnum)
: Evaluates to "1" if the response exists, and "0" if the response does not exist or was deleted.

$re_author(cnum inum rnum)
: Evaluates to the full name of the author (at the time the response was written).

$re_owner(cnum inum rnum)
: Evaluates to the userid of the author (owner) of the response.

$re_time(cnum inum rnum)
: Evaluates to the time and date the response was written.

$re_text(cnum inum rnum)
: Evaluates to the text of the response.

$re_prop(cnum inum rnum)
: Evaluates to the text property number of the response. (The property numbers are, for the moment, arbitrary, but are being used to distinguish how the user meant a response to be displayed -- e.g., as literal text with explicit line breaks, as plain text to be reformatted as simple HTML, or as explicit HTML as written by the responder.)

$re_title(cnum inum rnum)
: Evaluates to the title of the response. Only response 0 has a title, which is the title of the item.

$re_delete(cnum inum rnum)
: Deletes specified response. If *rnum* is 0, deletes entire item. Only the owner of the item or response (or an organizer) can successfully delete an item or response.

Note: as in section 4.14, all of these functions may be written with *no* arguments. In that case, the arguments from the previous use of any of these functions (in the same CML page) that *did* have arguments, are re-used.

4.16 Adding Items or Responses

$ad_resp(cnum prop inum text)

Adds *text* as a response to this item. Assumes the current user is the author. Evaluates to "1" if the adding succeeded, "0" if it failed. (Adding a response can fail if the user has read-only permission in the conference, or if the item is frozen.) Records *prop* as the text property number.

$ad_item(cnum prop title text)

Adds *text* as a new item. *Title* is the title of the new item. Note that there must be a newline between *title* and *text*. (See $newline(), section 4.3.) Assumes the current user is the author. Evaluates to the new item number if the adding succeeded, "0" if it failed. (Adding an item can fail if the organizer has turned off adding new items.) Records *prop* as the text property number.

$ad_author(name)

Sets the author of the next item or response to be added, to the psuedonymn *name*. This name will be used in place of the normal author name, in the next (and only the next) call to $ad_resp() or $ad_item().

$chg_resp(cnum prop inum rnum text)

Replaces the response (*cnum*, *inum*, *rnum*) with *text* and the new *prop* property value.

$chg_title(cnum prop inum title)

Changes the title of item (*cnum*, *inum*) to title. *Prop* is required, but ignored.

$set_wrap(width)

When text is added as an item or response, it is automatically column-wrapped before it is stored in the Caucus data files. Set_wrap sets the wrapping position to *width* single-width characters. A value of 0 turns column-wrapping off altogether.

4.17 Text Filters

$t2hbr(stuff)

Turns plain text *stuff* (which may contain newlines) into HTML. It turns each newline into a <BR>. It also turns each of the special characters <, ", and > into their HTML special codes (unless escaped by a "\"). Example:

```
" $t2hbr( shell(cat mytext) )
```

displays the text of an ordinary file *mytext* as HTML.

$safehtml(prop stuff)

"Safe HTML" filter. Filters HTML fragment in text of *stuff*, making it "safe" to include in an existing HTML page. Specifically, it removes the tags <HTML>, </HTML>, <HEAD>, </HEAD>, <BODY>, and </BODY>. It "closes" any open tags (such as <B>) that don't have a matching closing tag (such as </B>). It looks for mismatched quotes inside a tag, and adds an extra quote if necessary. (For example, <A HREF="junk> becomes <A HREF="junk">.)

Prop is a number that controls certain properties of $safehtml(). It is the sum of a set of bitmasks (powers of 2); each bit controls a particular property. The properties are:

1	allow <FORM>s. Otherwise <FORM> tags are removed, like <BODY>.

$rhtml(stuff)
Obsolete form of $safehtml(), without the Prop argument. $rhtml(stuff) is equivalent to $safehtml(0 stuff).

$t2html(stuff)
Attempts an "intelligent" filtering of plain text stuff into HTML. Blank lines become <P>'s, and short lines (except for ends of paragraphs) go inside <PRE> </PRE>. Parses and translates URL's into anchored links with the same names. (see $t2url().)

$t2url(stuff)
Translates URLs in *stuff* into anchored links with the same names. Both this function and $t2html() translate URLs that begin with any of the schemes http:/, gopher:/, telnet:/, ftp:/, or mailto:.

$wrap2html()
A more intelligent (than $t2html) filtering of plain text into HTML. Acts as much as possible like a typical word-processor. Each single “hard” RETURN in the original text translates into a <BR>; multiple RETURNs become sequences of “&nbsp;<P>”. Groups of N spaces become N-1 “&nbsp;”s plus a regular space. A tab is treated as a group of 5 spaces. Parses and translates URL’s into anchored links.

Special note: All 3 functions also recognize and translate special "caucus" URLs of the form "http:/caucus...", into a reference to a particular Caucus CML page on the current host (and with the current swebd subserver). For example, "http:/caucus" becomes a reference to the Caucus Center page, i.e. center.cml, and "http:/caucus/conf_name" becomes a reference to confhome.cml for conference *conf_name*. This is one of the **very** few instances in which the CML interpreter assumes knowledge of the names and arguments of the actual CML files. (Normally this would be a **bad** idea, but in this case the feature is so powerful and useful as to allow the exception.)

$t2amp(stuff)
Translates all "&"s in *stuff* into "&amp;". Useful to "pre-escape" HTML code that is going to be "unescaped" when displayed by a browser. (This pre-escaping is essential when using Caucus to edit a response containing HTML code. Without it, any escaped HTML special sequences like "&gt;" would lose their meaning after one edit.)

$escquote(text)
: Translates all double-quotes in *text* to the HTML special sequence “&quot;”. This is primarily useful for placing text (that contains double-quotes) inside a double-quote-delimited field inside an HTML <INPUT> tag.

$t2mail(address)
: Attempts to translate *address* into a "mailto:" URL. (For example, if *address* is "joe@xyz.com", $t2mail() generates "<a href="mailto:joe@xyz.com">joe@xyz.com</A>".) If *address* does not appear to be an e-mail address, it is passed through unchanged.

$wraptext(width text)
: Word-wraps *text* to *width* (single-width-character) columns by inserting newlines in the appropriate places.

4.18 User, Conference, and Item variables

The "regular" CML variables (e.g., "set var xyz" or "$(var)") are ephemeral: once the dedicated swebd server has exited, the values of those variables are lost.

CML also provides a set of variables that are persistent across sessions, and tied to a particular user, conference, or item. Such variables may contain any amount of text, including newlines. They provide a convenient way to extend a Caucus interface, and to customize how the interface appears to a particular user or in a particular conference or item.

Note that evaluating a variable is a fairly fast process. (All variables for a particular user, conference, or item, are loaded at once, and cached.) Setting a variable is much slower.

$user_var(user vname)
: Evaluates to the value of userid *user*'s variable called *vname*.

$set_user_var(user vname value)
: Sets userid *user*'s variable *vname* to *value*.

$conf_var(cnum vname)
: Evaluates to the value of conference *cnum*'s variable called *vname*.

$set_conf_var(cnum vname value)
: Sets conference *cnum*'s variable *vname* to *value*.

$item_var(cnum inum vname)
: Evaluates to the value of conference *cnum*, item *inum*'s variable called *vname*.

$set_item_var(cnum inum vname value)
: Sets conference *cnum*, item *inum*'s variable *vname* to *value*.

4.19 Searching Conference Text

Two very specialized functions provide the capability to search for and display text in the conference items and responses.

$find_it(cnum inum r0 r1 any inword text)

Search conference *cnum*, item *inum*, responses *r0* through *r1*. (If *r1* is -1, search through the last response). Look for the word (or words) in *text*.

The *any* and *inword* arguments modify exactly how and when the search succeeds. If *any* is 1, the search is successful if any of the words in text are found in a response. If *any* is 0, the search succeeds only if all of the words in text are found in the same response. If *inword* is 1, the words in text match no matter where they are found in the response -- including in the middle of a word in the response. (For example, "the" will match "o**the**r".) If inword is 0, matches must occur at the beginning of a word. (In that case, "the" will not match "other", but it will match "**the**sis".)

Find_it() evaluates to a triplet list of responses that had successful matches. (E.g., "17 2 5 17 2 8" means that responses 5 and 8 in item 2 in conference 17 had successful matches.)

$search_it(cnum inum r0 r1 any text)

This is an obsolete form of $find_it(). It is equivalent to $find_it() with an *inword* of 0.

$find_filter(size words... inword text)

Find_filter is really a text filter. It is meant to be used to display just the "hits" in a response that contains a word or words searched for via $search_it(). It boldfaces the searched-for words, and displays 3 lines of *text* around each hit.

Text is typically the entire text of a response. *Words* contains the word or words that were searched for. *Size* is the number of distinct words in *words*. *Inword* should have the same value it did in $find_it() -- it controls whether matches may be found in the middle of a word (*inword* = 1), or only at the beginning of a word (*inword* = 0).

$search_filter(size words... text)

This is an obsolete form of $find_filter(). It is equivalent to $find_filter() with an *inword* of 0.

4.20 CML Page Functions

One of the most challenging tasks in creating sophisticated interfaces in CML is keeping track of where the user has been. For example, a user may start at page A, go to page B to fill out a form, which in turn is processed by page C... which should return the user to page A. If page B may be invoked from many different places, this task (remembering where to return to after page C) can get quite complicated.

This issue is dealt with more fully in the forthcoming "Caucus 3.1 Programmer's Guide". This section details four CML functions which make this capability possible.

$page_save(refresh cmlfile arglist fragment description)

This function "saves" a CML page reference in a table inside the CML interpreter. It evaluates to (i.e., returns) a slot number in that table, which may be used by the other $page_... functions to access the saved page. The arguments to $page_save() define a page reference in such a way that the reference can be used later to easily "return to" that page later.

Cmlfile is the name of the CML file. *Arglist* is the list of arguments to that file that should be remembered. (*Arglist* must be one word, so typically the arguments are specified in their URL form, i.e. with plus signs separating the individual arguments.) *Fragment* is the anchor point where that document should be re-entered, e.g. "#here". (If there is no such anchor point, *fragment* should just be "#".) *Description* is just ordinary text that describes that page; it may be any number of words, including none.

The "Caucus Center" page shown in the example CML file in section 3 uses $page_save() to save the current location in a table slot:

```
set nxt $page_save (1 center.cml \
        $arg(2)+$arg(3)+$arg(4)+$arg(5)+$arg(6)+$arg(7)+$arg(8) \
             # $(center_name) )
```

This CML code fragment saves the current page (center.cml), with its list of arguments ($arg(2)+...), no fragment ("#"), and a text description (contained inside the variable center_name). The saved page reference is stored in a slot, and the slot number is stored (by the "set" statement) in variable **nxt**.

The *refresh* argument is somewhat complicated. The slot table in the CML interpreter has a fixed size... which means that slots that haven't been touched in a while will get automatically recycled. *Refresh* is a number that refers to the arguments in *arglist*. If *refresh* has a value of N, then the N'th argument in *arglist* is assumed to be a slot number, and that slot is refreshed -- that is, protected from being recycled until the rest of the slots in the table have been recycled. See the previously mentioned Programmer's Guide for more information.

$page_get(slot)
Evaluates to the entire string saved in *slot* (by $page_save()). The first word of the result is the page name, the second word is the argument list, the third word is the fragment (anchor name, with "#"), and the fourth through last words are the page description.

$page_return(slot #override empty)
Evaluates to a string that can be used in an HTTP "Location:" directive to "return to" a page saved in *slot*. *#override* is a fragment anchor that may be used to override the anchor that was saved (with $page_save()). If it is just "#", the original (saved) anchor is used, otherwise *#override* is used. *Empty* should be a full CML page reference, to be used only if there is no page saved at slot.

Here is an example from the Caucus 3.1 additemf.cml file, which processes adding a new item to a conference, and then returns to the page which invoked "create a new item":

```
"Location: $(href)/$page_return($arg(2) # center.cml?$(nch)+0+x+x+x+x+x+x)
```

In this case, $arg(2) is the slot number of the page that originally invoked "create a new item". There is no override on the saved fragment anchor, and the default page (in case there was no saved "calling" page) is center.cml, the "Caucus Center" page.

$page_caller(which slot)
Evaluates to the slot number of the page which "called" the page saved at *slot*. Assumes that the caller of a page is stored in the argument list to that page, in argument number *which*.

5. CML DIRECTIVES

CML pages are like mini-programs. They contain directives which control which lines of HTML code will actually get sent to the browser, or control how many times a set of HTML lines will be evaluated. There are ten directives, plus an "end" directive shared by "for", "count", "while", "if", "elif", and "else".

5.1 For

The CML "for" loop evaluates a set of lines multiple times. It looks like:

```
for variable1 [variable2 ... ] in list
   (HTML code or other CML directive code)
    ...
end
```

where *variable1, variable2* etc. are names, and *list* is a list of words or values. Typically *list* may be the result of a CML function. The for loop evaluates the lines between "for" and "end", substituting the words in *list* as the values of *variable1*, *variable2*, etc. (The brackets simply mean that *variable2*, etc. are optional. The brackets would not actually appear in the syntax of the for loop.)

For example, the loop:

```
for x in abc qrs xyz
    ...
end
```

will evaluate the lines between "for" and "end" three times, using each word in *list*. (If there are no words, the lines will be skipped.) The first time through the loop, **x** will have as its value "abc". The second time it will have the value "qrs", and so on.

A different example shows the use of multiple *variables*:

```
for one two in alpha beta delta gamma
   ...
end
```

The first time through the loop, **one** will have the value "alpha" and **two** will have the value "beta". The second time, **one** will have the value "delta", and so on.

The indenting of each line as shown above is not necessary, but it is a good idea. It helps make the CML code much more readable.

5.2 Count

The CML "count" loop is similar to the "for" loop. It looks like:

```
count variable x y
   (HTML code or other CML directive code)
    ...
end
```

where *variable* is a name, and *x* and *y* are numeric values or expressions. The count loop will evaluate the lines between "count" and "end" one time for each integer value between *x* and *y*, inclusive. The first time, variable will have the value *x*. Then *x*+1, and so on, up to and including *y*. If *y* is less than *x*, the lines will be skipped entirely.

5.3 While

The CML "while" loop is perhaps the simplest loop control directive. It has the form:

```
while expression
   (HTML code or other CML directive code)
    ...
end
```

The "while" loop evaluates *expression*, and examines the first word of the result. If it is a number, not equal to 0, all of the lines between "while" and "end" are evaluated. The loop then repeats, re-evaluating *expression*, and so on. The "while" loop will continue to execute as long as *expression* is non-zero, so be careful!

5.4 If

The CML "if" statement evaluates a set of lines if a certain condition is true. It looks like:

```
if condition
   (HTML code or other CML directive code)
    ...
end
```

where *condition* is some expression. If there is at least one word in *condition*, and the first word is a non-zero number, then the enclosed set of lines will be evaluated once. Otherwise they will be skipped. (Also see the related function $if() in section 4.6.)

5.5 Elif

The "if" statement may be extended to handle multiple exclusive cases with the "elif" directive. It looks like:

```
if condition1
   (HTML code or other CML directive code)
    ...
end
elif condition2
    ...
end
```

The lines between "elif" and "end" are evaluated when the previous "if" *condition1* failed (was 0 or did not exist) and the first word of *condition2* is a non-zero number.

Multiple "elif"s may be strung together, one after another. Only one of the blocks of CML code between the if/end and elif/end pairs will be executed.

5.6 Else

There is an (optional) matching "else" to the CML "if" and "elif" statements. It looks like:

```
if condition
   (HTML code or other CML directive code)
    ...
end
else
    ...
end
```

The lines between "else" and "end" are evaluated if *condition* is 0, or does not exist at all. *Note*: *the "if" must have its own "end"!* "Else" may be used with just an "if", or a series of "if"s and "elifs". If the latter, it must be the last of the series.

5.7 Set

The "for" and "count" directives define the value of a variable during iterations of the lines between the "for" or "count", and the matching "end" directive. Outside of those loops, the variable is undefined.

A variable may also be defined across the evaluation of all CML pages, using the "set" directive. It looks like:

```
set variable x
```

where *variable* is a name, and *x* is some expression. For the rest of this session, *variable* has the value *x* (unless changed by another "set" directive). Variables defined by "set" are considered "global" in scope, i.e. the variables are available in all subsequently evaluated CML pages.

5.8 Include

The "include" directive includes the text of a CML file at the current point. It has the syntax:

```
      include filename  [ arg1 [ arg2 ... ] ]
```

where *filename* is the name of a file, or a set of CML functions that evaluate to the name of a file. *Filename* is relative to the CML_Path directory specified in the swebd.conf file. (See the Caucus installation guide for details.) The brackets indicate that arguments *arg1*, *arg2*, and so on are optional (they are not actually part of the syntax). If the arguments are present, they are available inside the included file via the $inc(n) function (see section 4.2).

Include directives are evaluated according to the current context. For example:

```
      count x 1 3
         include file.$(x)
      end
```

would include the contents of the files **file.1**, **file.2**, and **file.3**.

5.9 Return

The "return" directive immediately ceases processing of the current CML file. It is particularly useful in CML pages that need to handle special case or "error" conditions. For example:

```
      if some "error" condition
         "Location: http://www.xyz.com/errorpage.html
         "
         return
      end

      #---OK, go on with the main case here...
      "Content-type: text/html
      "
      "etc...
```

5.10 Break

The "break" directive immediately exits the innermost "for", "count", or "while" loop, and continues execution of the CML script after the closing "end" of that loop.

Index to CML Functions