```
# $Id: ad-html.tcl,v 1.5 2006/05/13 18:38:21 philg Exp $
# /tcl/ad-html.tcl
#
# stuff for serving static .html pages
# (e.g., putting in comment links, etc.)

# written by philg@mit.edu on 7/1/98

# significantly enhanced in December 1999 to modularize the comment and link
# stuff so that .adp pages could use them as well (philg)

# any request for a static html file will go through this proc
# one good thing about doing things this way is that our site
# still looks static to AltaVista

# 10/21/2005: philg modified to include serving Google ads

# 11/5/2018: philg modified to use auto ads and Fifth Chance codes

if { ![ad_parameter "EnableAbstractURLsP" "abstract-url" 0] } {
    ad_register_proc GET /*.html ad_serve_html_page
    ad_register_proc GET /*.htm ad_serve_html_page
}

# this must stand for "do not disturb"
# if you put it into a static .html file, the
# following Tcl proc serves the page unmolested

proc ad_dnd_tag {} {
    return "<!--AD_DND-->"
}

proc ad_no_ads_tag {} {
    return "<!--AD_NOADS-->"
}

proc ad_insert_ads_tag {} {
    return "<!--AD_INSERTADS-->"
}

# these entire directories will have their .html
# files served intact

proc_doc ad_space { {n 1} } "returns n spaces in html (uses nbsp)" {
    set result ""
    for {set i 0} {$i < $n} {incr i} {
    append result " "
    }
    #append result " "
    return $result
}

proc ad_naked_html_patterns {} {
    set glob_patterns [list]
    lappend glob_patterns "/doc/*"
    lappend glob_patterns "/admin/*"
    lappend glob_patterns "/pages/*"
    lappend glob_patterns "/ct*"
    lappend glob_patterns "[ad_parameter GlobalURLStub "" "/global"]/*"
    return $glob_patterns
}

proc ad_google_ad_patterns {} {
    set glob_patterns [list]
    lappend glob_patterns "/materialism/*"
    lappend glob_patterns "/aquarium/*"
```

```
    lappend glob_patterns "/flying/*"
    lappend glob_patterns "/travel/*"
    lappend glob_patterns "/nz/*"
    lappend glob_patterns "/cr/*"
    lappend glob_patterns "/china/*"
    lappend glob_patterns "/book-reviews/*"
    lappend glob_patterns "/wtr/*"
    lappend glob_patterns "/wireless/*"
    return $glob_patterns
}

proc google_auto_ad {} {
    set code_from_google {<script async src=
    "//pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({
          google_ad_client: "ca-pub-6665459316689270",
          enable_page_level_ads: true
    });
    </script>}
    return $code_from_google
}

proc google_right_column_ad {} {
    set code_from_google {<script async src=
    "//pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- PhilipGreenspunRightColumn -->
<ins class="adsbygoogle"
    style="display:inline-block;width:160px;height:600px"
     data-ad-client="ca-pub-6665459316689270"
     data-ad-slot="6243994555"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>}
    return "<p class=rightgooglead>$code_from_google</p>\n"
}

proc google_center_bottom_ad {} {
    set code_from_google {<script async src=
    "//pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- PhilipGreenspunCenterBottom -->
<ins class="adsbygoogle"
    style="display:inline-block;width:728px;height:90px"
     data-ad-client="ca-pub-6665459316689270"
     data-ad-slot="1426782141"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>}
    return "<p class=bottomcentergooglead>$code_from_google</p>\n"
}

proc google_analytics_code {} {
    set code_from_google {<script type="text/javascript">

    var _gaq = _gaq || [];
    _gaq.push(['_setAccount', 'UA-315149-1']);
    _gaq.push(['_trackPageview']);

    (function() {
        var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async
        = true;
        ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www')
        + '.google-analytics.com/ga.js';
        var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga,
         s);
    })();
```

```
127
128   </script>}
129       return $code_from_google
130   }
131
132   # Amazon ads
133
134   # a tag inside the HTML file can expand to an amazon product display or possibly just a
      text link, whatever is more effective
135   # currently there is no single page on the Amazon Web site
136
137   # this will go into REGSUB so need to escape all of the & characters
138
139   proc amazon_canon_eos_big_block {} {
140       set code_from_amazon {
141   <p>
142   <center>
143   <iframe src=
      "http://rcm.amazon.com/e/cm?t=pgreenspun-20\&o=1\&p=16\&l=st1\&mode=photo\&search=canon%2
      0eos\&fc1=000000\&lt1=\&lc1=3366FF\&bg1=FFFFFF\&f=ifr" marginwidth="0" marginheight="0"
      width="468" height="336" border="0" frameborder="0" style="border:none;" scrolling="no"
      ></iframe>
144   </center>
145   </p>
146   }
147       return $code_from_amazon
148   }
149
150
151   proc_doc ad_serve_html_page {ignore} {The procedure that actually serves all the HTML
      pages on an ACS.  It looks first to see if the file is in one of the naked_html
      directories.  If so, it simply returns the raw bytes.  It then looks to see if the
      ad_dnd_tag ("do not disturb") comment pattern is present.  Again, if so, it simply
      returns.  Otherwise, the procedure tries to add comments and related links.  If the
      database is busy, it will simply add links to comments and related links.} {
152       set url_stub [ad_conn full_url]
153       if { [empty_string_p $url_stub] } {
154           set url_stub [ns_conn url]
155       }
156
157       set full_filename [ad_conn file]
158       if { [empty_string_p $full_filename] } {
159       set full_filename [ns_url2file $url_stub]
160       }
161
162       foreach naked_pattern [ad_naked_html_patterns] {
163       if [string match $naked_pattern $url_stub] {
164           ns_returnfile 200 text/html $full_filename
165           return
166       }
167       }
168
169       if { ![file exists $full_filename]} {
170       # check to see if the file exists
171       # if not, return a "file not found" message
172       set file_name_url "[ad_parameter GlobalURLStub "" "/global"]/file-not-found.html"
173       set full_path [ns_url2file $file_name_url]
174       if [file exists $full_path] {
175           ns_returnfile 404 text/html $full_path
176       } else {
177           ns_return 404 text/plain "File not found"
178       }
179       return
180       }
181
182       set stream [open $full_filename r]
```

```
183         set whole_page [read $stream]
184         close $stream
185 
186         ## sometimes we don't want comments to come up
187         ## for a given page
188         if {[string first [ad_dnd_tag] $whole_page] != -1} {
189         ns_return 200 text/html $whole_page
190         return
191         }
192 
193         # let's put in Amazon ads
194         regsub -all -nocase "<!--AMAZON_CANON_EOS_SYSTEM-->" $whole_page [
            amazon_canon_eos_big_block] whole_page
195 
196         # let's figure out if we're going to try to insert Google ads
197         set insert_google_ad_p 0
198         foreach google_ad_pattern [ad_google_ad_patterns] {
199         if [string match $google_ad_pattern $url_stub] {
200             set insert_google_ad_p 1
201             # matches at least one pattern, let's break out of the loop and stop wasting CPU
202             break
203         }
204         }
205         # let's check to see if this is the index file or a directory (ends in /)
206         set just_the_filename [file rootname [file tail $url_stub]]
207         if { $just_the_filename == "index" || [file tail $url_stub] == "" } {
208         # we don't want ads in the index files, which are carefully formatted
209         set insert_google_ad_p 0
210         }
211         if {[string first [ad_no_ads_tag] $whole_page] != -1} {
212         set insert_google_ad_p 0
213         }
214         if {[string first [ad_insert_ads_tag] $whole_page] != -1 } {
215         # whatever else happened, possibly on an index page, the publisher wants ads
216         set insert_google_ad_p 1
217         }
218 
219 
220         if { [regexp -nocase {(.*)</body>(.*)} $whole_page match pre_body post_body] } {
221         # there was a "close body" tag, let's try to insert a comment
222         # link at least
223         # before we do anything else, let's stream out what we can
224         if $insert_google_ad_p {
225             # let's try to insert a Google ad
226             # ns_log Notice "trying to insert a Google ad in $url_stub"
227             # we don't do "-all" so this should just be the first HR tag
228             regsub "<hr>" $pre_body "<hr>\n[google_right_column_ad]" pre_body
229             append pre_body "\n" [google_center_bottom_ad]
230         }
231         ad_return_top_of_page [static_add_curriculum_bar_if_necessary $pre_body]
232 
233         if { [catch { set db [ns_db gethandle -timeout -1] } errmsg] || [empty_string_p $db]
            } {
234             # the non-blocking call to gethandle raised a Tcl error; this
235             # means a db conn isn't free right this moment, so let's just
236             # return the page with a link
237             ns_log Notice "DB handle wasn't available in ad_serve_html_page"
238             ns_write "
239 <hr width=300>
240 <center>
241 <a href=\"/comments/for-one-page?url_stub=[ns_urlencode $url_stub]\">View/Add
    Comments</a> |
242 <a href=\"/links/for-one-page?url_stub=[ns_urlencode $url_stub]\">Related Links</a>
243 </center>
244 [google_analytics_code]
245 </body>$post_body"
```

```
246             } else {
247             # we got a db connection
248             set moby_list [static_get_comments_and_links $db $url_stub $post_body]
249             # Release the DB handle
250             ns_db releasehandle $db
251             set comment_link_options_fragment [static_format_comments_and_links $moby_list]
252             # now decide what to do with the comments and links we're queried from the
                database
253             ns_write "$comment_link_options_fragment\n\n[google_analytics_code]</body>$
                post_body"
254         }
255         } else {
256         # couldn't find a </body> tag
257         ns_return 200 text/html $whole_page
258         }
259     }
260
261     # helper proc for sticking in curriculum bar when necessary
262
263     proc_doc static_add_curriculum_bar_if_necessary {pre_body} "Returns the page, up to the
        close body tag, with a curriculum bar added if necessary" {
264         if { ![ad_parameter EnabledP curriculum 0] || ![ad_parameter StickInStaticPagesP
            curriculum 0] } {
265         return $pre_body
266         }
267         set curriculum_bar [curriculum_bar]
268         if [empty_string_p $curriculum_bar] {
269         # we are using the curriculum system but this user doesn't need a bar
270         return $pre_body
271         }
272         # let's look for a good place to stuff the bar
273         # rely on maximal matching in REGEXP
274         if { [regexp -nocase {(.*)<hr>(.*)} $pre_body match up_to_last_hr after_last_hr] } {
275         # we found at least one HR, let's make sure that it is indeed
276         # at the bottom of the page
277         if { [string length $up_to_last_hr] > [string length $after_last_hr] } {
278             # this is indeed probably the last
279             append pre_body_with_curriculum_bar $up_to_last_hr "\n<center>[curriculum_bar]
                </center>\n" "<HR>" $after_last_hr
280         } else {
281             # found an HR but probably it isn't the last one
282             append pre_body_with_curriculum_bar $pre_body "\n<center>[curriculum_bar]
                </center>\n"
283         }
284         } else {
285         append pre_body_with_curriculum_bar $pre_body "\n<center>[curriculum_bar]</center>\n"
286         }
287         return $pre_body_with_curriculum_bar
288     }
289
290     # helper proc for coming back with options, info, etc.
291
292     proc_doc static_get_comments_and_links {db url_stub {post_body ""}} "Returns a list of
        comment_bytes link_bytes options_list comment_option link_option or the empty string if
        this page isn't registered in the database" {
293         set user_id [ad_get_user_id]
294         set selection [ns_db 0or1row $db "select
            page_id,accept_comments_p,accept_links_p,inline_comments_p,inline_links_p from
            static_pages where url_stub = '[DoubleApos $url_stub]'"]
295         if { $selection == "" } {
296         # this page isn't registered in the database so we can't
297         # accept comments on it or anything
298         ns_log Notice "Someone grabbed $url_stub but we weren't able to offer a comment link
            because this page isn't registered in the db"
299         return ""
300         } else {
```

```
301         set_variables_after_query
302         set options_list [list]
303         set comment_bytes ""
304         if { $inline_comments_p == "t" } {
305             # we display comments in-line
306             set selection [ns_db select $db "select comments.comment_id, comments.page_id,
                comments.user_id as poster_user_id, users.first_names || ' ' || users.last_name
                as user_name, message, posting_time, html_p, client_file_name, file_type,
                original_width, original_height, caption
307             from static_pages sp, comments_not_deleted comments, users
308             where sp.page_id = comments.page_id
309             and comments.user_id = users.user_id
310             and comments.page_id = $page_id
311             and comments.comment_type = 'alternative_perspective'
312             order by posting_time"]
313             set at_least_one_comment_found_p 0
314             while { [ns_db getrow $db $selection] } {
315             set_variables_after_query
316             set at_least_one_comment_found_p 1
317             append comment_bytes "<blockquote>
318             [format_static_comment $comment_id $client_file_name $file_type $original_width $
                original_height $caption $message $html_p]
319             "
320             if { $user_id == $poster_user_id} {
321                 # the user wrote the message, so let him/her edit it
322                 append comment_bytes "   \[ <A
                    HREF=\"/comments/persistent-edit?comment_id=$comment_id\">edit your
                    comment</a> \]\n"
323             }
324
325             append comment_bytes "<br><br>\n
326   -- <A HREF=\"/shared/community-member?user_id=$poster_user_id\">$user_name</a>"
327             append comment_bytes ", [util_AnsiDatetoPrettyDate $posting_time]"
328             append comment_bytes "</blockquote>\n"
329             }
330         }
331         if { $accept_comments_p == "t" && $inline_comments_p == "t" } {
332             # we only display the option if we're inlining comments;
333             # we assume that if the comments aren't in line but are legal
334             # then the publisher has an explicit link
335             set comment_option "<a href=\"/comments/add?page_id=$page_id\">Add a comment</a>"
336             lappend options_list $comment_option
337         } else {
338             set comment_option ""
339         }
340
341         # links
342         set link_bytes ""
343         if { $inline_links_p == "t" } {
344             set selection [ns_db select $db "select links.page_id, links.user_id as
                poster_user_id, users.first_names || ' ' || users.last_name as user_name,
                links.link_title, links.link_description, links.url
345             from static_pages sp, links, users
346             where sp.page_id = links.page_id
347             and users.user_id = links.user_id
348             and links.page_id = $page_id
349             and status = 'live'
350             order by posting_time"]
351             while { [ns_db getrow $db $selection] } {
352             set_variables_after_query
353             append link_bytes "<li><a href=\"$url\" rel=\"nofollow\">$link_title</a>- $
                link_description"
354
355             if { $user_id == $poster_user_id} {
356                 # the user added, so let him/her edit it
357                 append link_bytes "  (<A HREF=\"/links/edit?page_id=$page_id&url=[
```

```
            ns_urlencode $url]\">edit/delete</a>)"
        } else {
            # the user did not add it, link to the community_member page
            append link_bytes "    <font size=-1>(contributed by <A
            HREF=\"/shared/community-member?user_id=$poster_user_id\">$
            user_name</a>)</font>"
        }
        append link_bytes "\n<p>\n"
        }
    }
    if { $accept_links_p == "t" && $inline_links_p == "t" } {
        # we only display the option if we're inlining links;
        # we assume that if the links aren't in line but are legal
        # then the publisher has an explicit link
        set link_option "<a href=\"/links/add?page_id=$page_id\">Add a link</a>"
        lappend options_list $link_option
    } else {
        set link_option ""
    }
    }
    return [list $comment_bytes $link_bytes $options_list $comment_option $link_option]
}


# helper proc for formatting comments, links, etc.

proc_doc static_format_comments_and_links {moby_list} "Takes list of comment_bytes
link_bytes options_list comment_option link_option and produces HTML fragment to stick
at bottom of page." {
    if [empty_string_p $moby_list] {
    return ""
    }
    set comment_bytes [lindex $moby_list 0]
    set link_bytes [lindex $moby_list 1]
    set options_list [lindex $moby_list 2]
    set comment_option [lindex $moby_list 3]
    set link_option [lindex $moby_list 4]
    if { [empty_string_p $comment_bytes] && [empty_string_p $link_bytes] } {
    if { [llength $options_list] > 0 } {
        set centered_options "<center>[join $options_list " | "]</center>"
    } else {
        set centered_options ""
    }
    return $centered_options
    } elseif { ![empty_string_p $comment_bytes] && [empty_string_p $link_bytes] } {
    # there are comments but no links
    return "<center><h3>Reader's Comments</h3></center>
    $comment_bytes
    <center>[join $options_list " | "]</center>"
    } elseif { [empty_string_p $comment_bytes] && ![empty_string_p $link_bytes] } {
    # links but no comments
    return "<center><h3>Related Links</h3></center>
    <ul>$link_bytes</ul>
    <center>[join $options_list " | "]</center>"
    } else {
    # comments and links
    return "<center><h3>Reader's Comments</h3></center>
    $comment_bytes
    <center>
    $comment_option
    </center>
    <center><h3>Related Links</h3></center>
    <ul>$link_bytes</ul>
    <center>
    $link_option
    </center>"
```

```
419         }
420     }
421
422     # Helper procedure for formatting 'alternative_perspective' comments on static
423     # pages, which presents inline images or attachment links as appropriate.
424     # Taken from a similar procedure in ad-general-comments.tcl.
425     proc format_static_comment { comment_id client_file_name file_type original_width
        original_height caption content comment_html_p } {
426         set return_string ""
427         set return_url "[ns_conn url]?[export_ns_set_vars url]"
428
429         if { ![empty_string_p $client_file_name] } {
430         # We have an attachment.
431         if { [string match "image/*" [string tolower $file_type]] } {
432             # It was an image.
433             if { ![empty_string_p $original_width]
434              && $original_width < [ad_parameter InlineImageMaxWidth "comments" 512] } {
435             # It's narrow enough to display inline.
436             append return_string "<center><img src=\"/comments/attachment/$comment_id/$
                client_file_name\" width=$original_width height=$original_height><p><i>$
                caption</i></center><br>\n[util_maybe_convert_to_html $content $comment_html_p]
                \n"
437             } else {
438             # Send to an image display page.
439             append return_string "[util_maybe_convert_to_html $content $comment_html_p]
                \n<br><i>Image: <a href=\"/comments/image-attachment?[export_url_vars comment_id
                return_url]\">$client_file_name</a></i>"
440             }
441         } else {
442             # Send to raw file download.
443             append return_string "[util_maybe_convert_to_html $content $comment_html_p]
                \n<br><i>Attachment: <a href=\"/comments/attachment/$comment_id/$
                client_file_name\">$client_file_name</a></i>"
444         }
445         } else {
446         # No attachment
447         append return_string "[util_maybe_convert_to_html $content $comment_html_p]\n"
448         }
449         return $return_string
450     }
451
452
453     proc_doc send_author_comment_p { comment_type action } "Returns email notification state
        type of html comment" {
454
455         if { [string compare $action "add"] == 0 } {
456
457         switch $comment_type {
458             "unanswered_question" { return [ad_parameter EmailNewUnansweredQuestion comments]
                 }
459             "alternative_perspective" { return [ad_parameter EmailNewAlternativePerspective
                comments] }
460             "rating" { return [ad_parameter EmailNewRating comments] }
461             default  { return 0 }
462         }
463
464         } else {
465
466         switch $comment_type {
467            "unanswered_question" { return [ad_parameter EmailEditedUnansweredQuestion
               comments] }
468            "alternative_perspective" { return [ad_parameter EmailEditedAlternativePerspective
                comments] }
469            "rating" { return [ad_parameter EmailEditedRating comments] }
470             default  { return 0 }
471         }
```

```
472         }
473     }
474
475
476     # ns_register'ed to
477     # /comments/attachment/[comment_id]/[file_name] Returns a
478     # MIME-typed attachment based on the comment_id. We use this so that
479     # the user's browser shows the filename the file was uploaded with
480     # when prompting to save instead of the name of a Tcl file (like
481     # "raw-file.tcl")
482     # Stolen from ad-general-comments.tcl.
483
484     proc ad_static_comments_get_attachment { ignore } {
485         if { ![regexp {([^/]+)/([^/]+)$} [ns_conn url] match comment_id client_filename] } {
486         ad_return_error "Malformed Attachment Request" "Your request for a file attachment
        was malformed."
487         return
488         }
489         set db [ns_db gethandle subquery]
490
491
492         # make sure comment_id is an integer
493         validate_integer "comment_id" $comment_id
494
495         set file_type [database_to_tcl_string $db "select file_type
496         from comments
497         where comment_id = $comment_id"]
498
499         # this is the new stuff
500         # written by dan parker
501         set tmp_file_name [ns_mktemp /web/philip/tmp_files/${comment_id}XXXXXX]
502
503         ns_ora blob_get_file $db "
504         select attachment
505         from comments
506         where comment_id = $comment_id" $tmp_file_name
507
508         ns_db releasehandle $db
509
510         ns_returnfile 200 $file_type $tmp_file_name
511
512         ns_unlink $tmp_file_name
513         # end of new stuff
514
515         # old stuff to eb commented out
516
517     #    ReturnHeaders $file_type
518
519     #    if [catch { ns_ora write_blob $db "select attachment
520     #    from comments
521     #    where comment_id = $comment_id"} errmsg] {
522     #   set url [ns_conn url]
523     #   set header [ns_conn headers]
524     #   set user_agent [ns_set iget $header USER-AGENT]
525     #   set referer [ns_set iget $header REFERER]
526     #   set peeraddr [peeraddr]
527     #   ns_log warning "dan_error 5567 $comment_id url = $url -  perraddr = $peeraddr -
    referer = $referer -  user_agent = $user_agent
528     #"
529     #    }
530     #    ns_db releasehandle $db
531     }
532
533     ad_register_proc GET /comments/attachment/* ad_static_comments_get_attachment
534
535
```