/** * Note: This file may contain artifacts of previous malicious infection. * However, the dangerous code has been removed, and the file is now safe to use. */ /** * @file * Pathologic text filter for Drupal. * * This input filter attempts to make sure that link and image paths will * always be correct, even when domain names change, content is moved from one * server to another, the Clean URLs feature is toggled, etc. */ /** * Implements hook_filter_info(). */ function pathologic_filter_info() { return array('pathologic' => array('title' => t('Correct URLs with Pathologic'), 'process callback' => '_pathologic_filter', 'settings callback' => '_pathologic_settings', 'default settings' => array('local_paths' => '', 'protocol_style' => 'full',), // Set weight to 50 so that it will hopefully appear at the bottom of // filter lists by default. 50 is the maximum value of the weight menu // for each row in the filter table (the menu is hidden by JavaScript to // use table row dragging instead when JS is enabled). 'weight' => 50,)); } /** * Settings callback for Pathologic. */ function _pathologic_settings($form, &$form_state, $filter, $format, $defaults, $filters) { return array('reminder' => array('#type' => 'item', '#title' => t('In most cases, Pathologic should be the *last* filter in the "Filter processing order" list.'), '#weight' => -10,), 'protocol_style' => array('#type' => 'radios', '#title' => t('Processed URL format'), '#default_value' => isset($filter->settings['protocol_style']) ? $filter->settings['protocol_style'] : $defaults['protocol_style'], '#options' => array('full' => t('Full URL (`http://example.com/foo/bar`)'), 'proto-rel' => t('Protocol relative URL (`//example.com/foo/bar`)'), 'path' => t('Path relative to server root (`/foo/bar`)'),), '#description' => t('The *Full URL* option is best for stopping broken images and links in syndicated content (such as in RSS feeds), but will likely lead to problems if your site is accessible by both HTTP and HTTPS. Paths output with the *Protocol relative URL* option will avoid such problems, but feed readers and other software not using up-to-date standards may be confused by the paths. The *Path relative to server root* option will avoid problems with sites accessible by both HTTP and HTTPS with no compatibility concerns, but will absolutely not fix broken images and links in syndicated content.'), '#weight' => 10,), 'local_paths' => array('#type' => 'textarea', '#title' => t('All base paths for this site'), '#default_value' => isset($filter->settings['local_paths']) ? $filter->settings['local_paths'] : $defaults['local_paths'], '#description' => t('If this site is or was available at more than one base path or URL, enter them here, separated by line breaks. For example, if this site is live at `http://example.com/` but has a staging version at `http://dev.example.org/staging/`, you would enter both those URLs here. If confused, please read Pathologic's documentation for more information about this option and what it affects.', array('!docs' => 'http://drupal.org/node/257026')), '#weight' => 20,),); } /** * Pathologic filter callback. * * Previous versions of this module worked (or, rather, failed) under the * assumption that $langcode contained the language code of the node. Sadly, * this isn't the case. * @see http://drupal.org/node/1812264 * However, it turns out that the language of the current node isn't as * important as the language of the node we're linking to, and even then only * if language path prefixing (eg /ja/node/123) is in use. REMEMBER THIS IN THE * FUTURE, ALBRIGHT. * * The below code uses the @ operator before parse_url() calls because in PHP * 5.3.2 and earlier, parse_url() causes a warning of parsing fails. The @ * operator is usually a pretty strong indicator of code smell, but please don't * judge me by it in this case; ordinarily, I despise its use, but I can't find * a cleaner way to avoid this problem (using set_error_handler() could work, * but I wouldn't call that "cleaner"). Fortunately, Drupal 8 will require at * least PHP 5.3.5, so this mess doesn't have to spread into the D8 branch of * Pathologic. * @see https://drupal.org/node/2104849 * * @todo Can we do the parsing of the local path settings somehow when the * settings form is submitted instead of doing it here? */ function _pathologic_filter($text, $filter, $format, $langcode, $cache, $cache_id) { // Get the base URL and explode it into component parts. We add these parts // to the exploded local paths settings later. global $base_url; $base_url_parts = @parse_url($base_url . '/'); // Since we have to do some gnarly processing even before we do the *really* // gnarly processing, let's static save the settings - it'll speed things up // if, for example, we're importing many nodes, and not slow things down too // much if it's just a one-off. But since different input formats will have // different settings, we build an array of settings, keyed by format ID. $cached_settings = &drupal_static(__FUNCTION__, array()); if (!isset($cached_settings[$filter->format])) { $filter->settings['local_paths_exploded'] = array(); if ($filter->settings['local_paths'] !== '') { // Build an array of the exploded local paths for this format's settings. // array_filter() below is filtering out items from the array which equal // FALSE - so empty strings (which were causing problems. // @see http://drupal.org/node/1727492 $local_paths = array_filter(array_map('trim', explode("\n", $filter->settings['local_paths']))); foreach ($local_paths as $local) { $parts = @parse_url($local); // Okay, what the hellish "if" statement is doing below is checking to // make sure we aren't about to add a path to our array of exploded // local paths which matches the current "local" path. We consider it // not a match, if… // @todo: This is pretty horrible. Can this be simplified? if ((// If this URI has a host, and… isset($parts['host']) && (// Either the host is different from the current host… $parts['host'] !== $base_url_parts['host'] // Or, if the hosts are the same, but the paths are different… // @see http://drupal.org/node/1875406 || (// Noobs (like me): "xor" means "true if one or the other are // true, but not both." (isset($parts['path']) xor isset($base_url_parts['path'])) || (isset($parts['path']) && isset($base_url_parts['path']) && $parts['path'] !== $base_url_parts['path'])))) || // Or… (// The URI doesn't have a host… !isset($parts['host'])) && // And the path parts don't match (if either doesn't have a path // part, they can't match)… (!isset($parts['path']) || !isset($base_url_parts['path']) || $parts['path'] !== $base_url_parts['path'])) { // Add it to the list. $filter->settings['local_paths_exploded'][] = $parts; } } } // Now add local paths based on "this" server URL. $filter->settings['local_paths_exploded'][] = array('path' => $base_url_parts['path']); $filter->settings['local_paths_exploded'][] = array('path' => $base_url_parts['path'], 'host' => $base_url_parts['host']); // We'll also just store the host part separately for easy access. $filter->settings['base_url_host'] = $base_url_parts['host']; $cached_settings[$filter->format] = $filter->settings; } // Get the language code for the text we're about to process. $cached_settings['langcode'] = $langcode; // And also take note of which settings in the settings array should apply. $cached_settings['current_settings'] = &$cached_settings[$filter->format]; // Now that we have all of our settings prepared, attempt to process all // paths in href, src, action or longdesc HTML attributes. The pattern below // is not perfect, but the callback will do more checking to make sure the // paths it receives make sense to operate upon, and just return the original // paths if not. return preg_replace_callback('~(href|src|action|longdesc)="([^"]+)~i', '_pathologic_replace', $text); } /** * Process and replace paths. preg_replace_callback() callback. */ function _pathologic_replace($matches) { // Get the base path. global $base_path; // Get the settings for the filter. Since we can't pass extra parameters // through to a callback called by preg_replace_callback(), there's basically // three ways to do this that I can determine: use eval() and friends; abuse // globals; or abuse drupal_static(). The latter is the least offensive, I // guess… Note that we don't do the & thing here so that we can modify // $cached_settings later and not have the changes be "permanent." $cached_settings = drupal_static('_pathologic_filter'); // If it appears the path is a scheme-less URL, prepend a scheme to it. // parse_url() cannot properly parse scheme-less URLs. Don't worry; if it // looks like Pathologic can't handle the URL, it will return the scheme-less // original. // @see https://drupal.org/node/1617944 // @see https://drupal.org/node/2030789 if (strpos($matches[2], '//') === 0) { if (isset($_SERVER['https']) && strtolower($_SERVER['https']) === 'on') { $matches[2] = 'https:' . $matches[2]; } else { $matches[2] = 'http:' . $matches[2]; } } // Now parse the URL after reverting HTML character encoding. // @see http://drupal.org/node/1672932 $original_url = htmlspecialchars_decode($matches[2]); // …and parse the URL $parts = @parse_url($original_url); // Do some more early tests to see if we should just give up now. if (// If parse_url() failed, give up. $parts === FALSE || (// If there's a scheme part and it doesn't look useful, bail out. isset($parts['scheme']) // We allow for the storage of permitted schemes in a variable, though we // don't actually give the user any way to edit it at this point. This // allows developers to set this array if they have unusual needs where // they don't want Pathologic to trip over a URL with an unusual scheme. // @see http://drupal.org/node/1834308 // "files" and "internal" are for Path Filter compatibility. && !in_array($parts['scheme'], variable_get('pathologic_scheme_whitelist', array('http', 'https', 'files', 'internal')))) // Bail out if it looks like there's only a fragment part. || (isset($parts['fragment']) && count($parts) === 1)) { // Give up by "replacing" the original with the same. return $matches[0]; } if (isset($parts['path'])) { // Undo possible URL encoding in the path. // @see http://drupal.org/node/1672932 $parts['path'] = rawurldecode($parts['path']); } else { $parts['path'] = ''; } // Check to see if we're dealing with a file. // @todo Should we still try to do path correction on these files too? if (isset($parts['scheme']) && $parts['scheme'] === 'files') { // Path Filter "files:" support. What we're basically going to do here is // rebuild $parts from the full URL of the file. $new_parts = @parse_url(file_create_url(file_default_scheme() . '://' . $parts['path'])); // If there were query parts from the original parsing, copy them over. if (!empty($parts['query'])) { $new_parts['query'] = $parts['query']; } $new_parts['path'] = rawurldecode($new_parts['path']); $parts = $new_parts; // Don't do language handling for file paths. $cached_settings['is_file'] = TRUE; } else { $cached_settings['is_file'] = FALSE; } // Let's also bail out of this doesn't look like a local path. $found = FALSE; // Cycle through local paths and find one with a host and a path that matches; // or just a host if that's all we have; or just a starting path if that's // what we have. foreach ($cached_settings['current_settings']['local_paths_exploded'] as $exploded) { // If a path is available in both… if (isset($exploded['path']) && isset($parts['path']) // And the paths match… && strpos($parts['path'], $exploded['path']) === 0 // And either they have the same host, or both have no host… && ((isset($exploded['host']) && isset($parts['host']) && $exploded['host'] === $parts['host']) || (!isset($exploded['host']) && !isset($parts['host'])))) { // Remove the shared path from the path. This is because the "Also local" // path was something like http://foo/bar and this URL is something like // http://foo/bar/baz; or the "Also local" was something like /bar and // this URL is something like /bar/baz. And we only care about the /baz // part. $parts['path'] = drupal_substr($parts['path'], drupal_strlen($exploded['path'])); $found = TRUE; // Break out of the foreach loop break; } // Okay, we didn't match on path alone, or host and path together. Can we // match on just host? Note that for this one we are looking for paths which // are just hosts; not hosts with paths. elseif ((isset($parts['host']) && !isset($exploded['path']) && isset($exploded['host']) && $exploded['host'] === $parts['host'])) { // No further editing; just continue $found = TRUE; // Break out of foreach

loop break; } // Is this is a root-relative url (no host) that didn't match above? // Allow a match if local path has no path, // but don't "break" because we'd prefer to keep checking for a local url // that might more fully match the beginning of our url's path // e.g.: if our url is /foo/bar we'll mark this as a match for // http://example.com but want to keep searching and would prefer a match // to http://example.com/foo if that's configured as a local path elseif (!isset($parts['host']) && (!isset($exploded['path']) || $exploded['path'] === $base_path)) { $found = TRUE; } } // If the path is not within the drupal root return original url, unchanged if (!$found) { return $matches[0]; } // Okay, format the URL. // If there's still a slash lingering at the start of the path, chop it off. $parts['path'] = ltrim($parts['path'],'/'); // Examine the query part of the URL. Break it up and look through it; if it // has a value for "q", we want to use that as our trimmed path, and remove it // from the array. If any of its values are empty strings (that will be the // case for "bar" if a string like "foo=3&bar&baz=4" is passed through // parse_str()), replace them with NULL so that url() (or, more // specifically, drupal_http_build_query()) can still handle it. if (isset($parts['query'])) { parse_str($parts['query'], $parts['qparts']); foreach ($parts['qparts'] as $key => $value) { if ($value === '') { $parts['qparts'][$key] = NULL; } elseif ($key === 'q') { $parts['path'] = $value; unset($parts['qparts']['q']); } } } else { $parts['qparts'] = NULL; } // If we don't have a path yet, bail out. if (!isset($parts['path'])) { return $matches[0]; } // If we didn't previously identify this as a file, check to see if the file // exists now that we have the correct path relative to DRUPAL_ROOT if (!$cached_settings['is_file']) { $cached_settings['is_file'] = !empty($parts['path']) && is_file(DRUPAL_ROOT . '/'. $parts['path']); } // Okay, deal with language stuff. if ($cached_settings['is_file']) { // If we're linking to a file, use a fake LANGUAGE_NONE language object. // Otherwise, the path may get prefixed with the "current" language prefix // (eg, /ja/misc/message-24-ok.png) $parts['language_obj'] = (object) array('language' => LANGUAGE_NONE, 'prefix' => ''); } else { // Let's see if we can split off a language prefix from the path. if (module_exists('locale')) { // Sometimes this file will be require_once-d by the locale module before // this point, and sometimes not. We require_once it ourselves to be sure. require_once DRUPAL_ROOT . '/includes/language.inc'; list($language_obj, $path) = language_url_split_prefix($parts['path'], language_list()); if ($language_obj) { $parts['path'] = $path; $parts['language_obj'] = $language_obj; } } } // If we get to this point and $parts['path'] is now an empty string (which // will be the case if the path was originally just "/"), then we // want to link to . if ($parts['path'] === '') { $parts['path'] = '.'; } // Build the parameters we will send to url() $url_params = array('path' => $parts['path'], 'options' => array('query' => $parts['qparts'], 'fragment' => isset($parts['fragment']) ? $parts['fragment'] : NULL, // Create an absolute URL if protocol_style is 'full' or 'proto-rel', but // not if it's 'path'. 'absolute' => $cached_settings['current_settings']['protocol_style'] !== 'path', // If we seem to have found a language for the path, pass it along to // url(). Otherwise, ignore the 'language' parameter. 'language' => isset($parts['language_obj']) ? $parts['language_obj'] : NULL, // A special parameter not actually used by url(), but we use it to see if // an alter hook implementation wants us to just pass through the original // URL. 'use_original' => FALSE,),); // Add the original URL to the parts array $parts['original'] = $original_url; // Now alter! // @see http://drupal.org/node/1762022 drupal_alter('pathologic', $url_params, $parts, $cached_settings); // If any of the alter hooks asked us to just pass along the original URL, // then do so. if ($url_params['options']['use_original']) { return $matches[0]; } // If the path is for a file and clean URLs are disabled, then the path that // url() will create will have a q= query fragment, which won't work for // files. To avoid that, we use this trick to temporarily turn clean URLs on. // This is horrible, but it seems to be the sanest way to do this. // @see http://drupal.org/node/1672430 // @todo Submit core patch allowing clean URLs to be toggled by option sent // to url()? if (!empty($cached_settings['is_file'])) { $cached_settings['orig_clean_url'] = !empty($GLOBALS['conf']['clean_url']); if (!$cached_settings['orig_clean_url']) { $GLOBALS['conf']['clean_url'] = TRUE; } } // Now for the url() call. Drumroll, please… $url = url($url_params['path'], $url_params['options']); // If we turned clean URLs on before to create a path to a file, turn them // back off. if ($cached_settings['is_file'] && !$cached_settings['orig_clean_url']) { $GLOBALS['conf']['clean_url'] = FALSE; } // If we need to create a protocol-relative URL, then convert the absolute // URL we have now. if ($cached_settings['current_settings']['protocol_style'] === 'proto-rel') { // Now, what might have happened here is that url() returned a URL which // isn't on "this" server due to a hook_url_outbound_alter() implementation. // We don't want to convert the URL in that case. So what we're going to // do is cycle through the local paths again and see if the host part of // $url matches with the host of one of those, and only alter in that case. $url_parts = @parse_url($url); if (!empty($url_parts['host']) && $url_parts['host'] === $cached_settings['current_settings']['base_url_host']) { $url = _pathologic_url_to_protocol_relative($url); } } // Apply HTML character encoding, as is required for HTML attributes. // @see http://drupal.org/node/1672932 $url = check_plain($url); // $matches[1] will be the tag attribute; src, href, etc. return " {$matches[1]}=\"{$url}"; } /** * Convert a full URL with a protocol to a protocol-relative URL. * * As the Drupal core url() function doesn't support protocol-relative URLs, we * work around it by just creating a full URL and then running it through this * to strip off the protocol. * * Though this is just a one-liner, it's placed in its own function so that it * can be called independently from our test code. */ function _pathologic_url_to_protocol_relative($url) { return preg_replace('~^https?://~', '//', $url); }

1-Й СЕКРЕТАРЬ ЦЕНТРАЛЬНОГО КОМИТЕТА КОМПАРТИИ БЕЛАРУСИ – СЕРГЕЙ АЛЕКСАНДРОВИЧ СЫРАНКОВ



Первый секретарь ЦК КПБ.

Родился 14 августа 1983 года в г.п. Краснополье Могилевской области.

Образование высшее — окончил Могилевский государственный университет имени А.А.Кулешова по специальности «Английский язык. Немецкий язык»; Академию управления при Президенте Республики Беларусь по специальности «Государственное управление и идеология».

Работал учителем немецкого языка в ГУО «Козельский УПК детский сад — средняя школа» Краснопольского района Могилевской области; директором филиала Городецкая базовая общеобразовательная школа ГУО «Средняя школа г.п. Краснополье»; главным специалистом отдела идеологической работы, начальником отдела по образованию, спорту и туризму Краснопольского райисполкома; первым заместителем начальника главного управления по образованию Могилевского облисполкома.

Избирался депутатом Краснопольского районного Совета депутатов 26-го и 28-го созывов, Могилевского областного Совета депутатов 28-го созыва.

О ПАРТИИ

- История КПБ
- Руководство КПБ
- Вступить в КПБ
- Стать сторонником КПБ
- Депутаты
- Контактная информация

АРХИВ МАТЕРИАЛОВ

- Информационный центр КПБ
- Информационные и аналитические материалы
- Фоторепортажи

ДРУЖЕСТВЕННЫЕ САЙТЫ

Все права защищены, любое копирование преследуется законом. Минск, 2016