[Overview](http://docs.google.com/index.html)

### Packages

* [CodeIgniter](http://docs.google.com/package-CodeIgniter.html)
  + [Libraries](http://docs.google.com/package-CodeIgniter.Libraries.html)
  + [Rest](http://docs.google.com/package-CodeIgniter.Rest.html)
* [None](http://docs.google.com/package-None.html)

### Classes

* [Example](http://docs.google.com/class-Example.html)
* [Format](http://docs.google.com/class-Format.html)
* [Key](http://docs.google.com/class-Key.html)
* [REST\_Controller](http://docs.google.com/class-REST_Controller.html)
* [Rest\_server](http://docs.google.com/class-Rest_server.html)
* [Welcome](http://docs.google.com/class-Welcome.html)
* [Overview](http://docs.google.com/index.html)
* Package
* Class

[1:](#gjdgxs) <?php  
 [2:](#30j0zll)   
 [3:](#1fob9te) defined('BASEPATH') OR exit('No direct script access allowed');  
 [4:](#3znysh7)   
 [5:](#2et92p0) /\*\*  
 [6:](#tyjcwt)  \* Format class  
 [7:](#3dy6vkm)  \* Help convert between various formats such as XML, JSON, CSV, etc.  
 [8:](#1t3h5sf)  \*  
 [9:](#4d34og8)  \* @author Phil Sturgeon, Chris Kacerguis, @softwarespot  
 [10:](#2s8eyo1)  \* @license http://www.dbad-license.org/  
 [11:](#17dp8vu)  \*/  
 [12:](#3rdcrjn) class Format {  
 [13:](#26in1rg)   
 [14:](#lnxbz9)  /\*\*  
 [15:](#35nkun2)  \* Array output format  
 [16:](#1ksv4uv)  \*/  
 [17:](#44sinio)  const ARRAY\_FORMAT = 'array';  
 [18:](#2jxsxqh)   
 [19:](#z337ya)  /\*\*  
 [20:](#3j2qqm3)  \* Comma Separated Value (CSV) output format  
 [21:](#1y810tw)  \*/  
 [22:](#4i7ojhp)  const CSV\_FORMAT = 'csv';  
 [23:](#2xcytpi)   
 [24:](#1ci93xb)  /\*\*  
 [25:](#3whwml4)  \* Json output format  
 [26:](#2bn6wsx)  \*/  
 [27:](#qsh70q)  const JSON\_FORMAT = 'json';  
 [28:](#3as4poj)   
 [29:](#1pxezwc)  /\*\*  
 [30:](#49x2ik5)  \* HTML output format  
 [31:](#2p2csry)  \*/  
 [32:](#147n2zr)  const HTML\_FORMAT = 'html';  
 [33:](#3o7alnk)   
 [34:](#23ckvvd)  /\*\*  
 [35:](#ihv636)  \* PHP output format  
 [36:](#32hioqz)  \*/  
 [37:](#1hmsyys)  const PHP\_FORMAT = 'php';  
 [38:](#41mghml)   
 [39:](#2grqrue)  /\*\*  
 [40:](#vx1227)  \* Serialized output format  
 [41:](#3fwokq0)  \*/  
 [42:](#1v1yuxt)  const SERIALIZED\_FORMAT = 'serialized';  
 [43:](#4f1mdlm)   
 [44:](#2u6wntf)  /\*\*  
 [45:](#19c6y18)  \* XML output format  
 [46:](#3tbugp1)  \*/  
 [47:](#28h4qwu)  const XML\_FORMAT = 'xml';  
 [48:](#nmf14n)   
 [49:](#37m2jsg)  /\*\*  
 [50:](#1mrcu09)  \* Default format of this class  
 [51:](#46r0co2)  \*/  
 [52:](#2lwamvv)  const DEFAULT\_FORMAT = self::JSON\_FORMAT; // Couldn't be DEFAULT, as this is a keyword  
 [53:](#111kx3o)   
 [54:](#3l18frh)  /\*\*  
 [55:](#206ipza)  \* CodeIgniter instance  
 [56:](#4k668n3)  \*  
 [57:](#2zbgiuw)  \* @var object  
 [58:](#1egqt2p)  \*/  
 [59:](#3ygebqi)  private $\_CI;  
 [60:](#2dlolyb)   
 [61:](#sqyw64)  /\*\*  
 [62:](#3cqmetx)  \* Data to parse  
 [63:](#1rvwp1q)  \*  
 [64:](#4bvk7pj)  \* @var mixed  
 [65:](#2r0uhxc)  \*/  
 [66:](#1664s55)  protected $\_data = [];  
 [67:](#3q5sasy)   
 [68:](#25b2l0r)  /\*\*  
 [69:](#kgcv8k)  \* Type to convert from  
 [70:](#34g0dwd)  \*  
 [71:](#1jlao46)  \* @var string  
 [72:](#43ky6rz)  \*/  
 [73:](#2iq8gzs)  protected $\_from\_type = NULL;  
 [74:](#xvir7l)   
 [75:](#3hv69ve)  /\*\*  
 [76:](#1x0gk37)  \* DO NOT CALL THIS DIRECTLY, USE factory()  
 [77:](#4h042r0)  \*  
 [78:](#2w5ecyt)  \* @param NULL $data  
 [79:](#1baon6m)  \* @param NULL $from\_type  
 [80:](#3vac5uf)  \* @throws Exception  
 [81:](#2afmg28)  \*/  
 [82:](#pkwqa1)   
 [83:](#39kk8xu)  public function \_\_construct($data = NULL, $from\_type = NULL)  
 [84:](#1opuj5n)  {  
 [85:](#48pi1tg)  // Get the CodeIgniter reference  
 [86:](#2nusc19)  $this->\_CI = &get\_instance();  
 [87:](#1302m92)   
 [88:](#3mzq4wv)  // Load the inflector helper  
 [89:](#2250f4o)  $this->\_CI->load->helper('inflector');  
 [90:](#haapch)   
 [91:](#319y80a)  // If the provided data is already formatted we should probably convert it to an array  
 [92:](#1gf8i83)  if ($from\_type !== NULL)  
 [93:](#40ew0vw)  {  
 [94:](#2fk6b3p)  if (method\_exists($this, '\_from\_'.$from\_type))  
 [95:](#upglbi)  {  
 [96:](#3ep43zb)  $data = call\_user\_func([$this, '\_from\_'.$from\_type], $data);  
 [97:](#1tuee74)  }  
 [98:](#4du1wux)  else  
 [99:](#2szc72q)  {  
[100:](#184mhaj)  throw new Exception('Format class does not support conversion from "'.$from\_type.'".');  
[101:](#3s49zyc)  }  
[102:](#279ka65)  }  
[103:](#meukdy)   
[104:](#36ei31r)  // Set the member variable to the data passed  
[105:](#1ljsd9k)  $this->\_data = $data;  
[106:](#45jfvxd)  }  
[107:](#2koq656)   
[108:](#zu0gcz)  /\*\*  
[109:](#3jtnz0s)  \* Create an instance of the format class  
[110:](#1yyy98l)  \* e.g: echo $this->format->factory(['foo' => 'bar'])->to\_csv();  
[111:](#4iylrwe)  \*  
[112:](#2y3w247)  \* @param mixed $data Data to convert/parse  
[113:](#1d96cc0)  \* @param string $from\_type Type to convert from e.g. json, csv, html  
[114:](#3x8tuzt)  \*  
[115:](#2ce457m)  \* @return object Instance of the format class  
[116:](#rjefff)  \*/  
[117:](#3bj1y38)  public function factory($data, $from\_type = NULL)  
[118:](#1qoc8b1)  {  
[119:](#4anzqyu)  // $class = \_\_CLASS\_\_;  
[120:](#2pta16n)  // return new $class();  
[121:](#14ykbeg)   
[122:](#3oy7u29)  return new static($data, $from\_type);  
[123:](#243i4a2)  }  
[124:](#j8sehv)   
[125:](#338fx5o)  // FORMATTING OUTPUT ---------------------------------------------------------  
[126:](#1idq7dh)   
[127:](#42ddq1a)  /\*\*  
[128:](#2hio093)  \* Format data as an array  
[129:](#wnyagw)  \*  
[130:](#3gnlt4p)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[131:](#1vsw3ci)  \* to the constructor  
[132:](#4fsjm0b)  \* @return array Data parsed as an array; otherwise, an empty array  
[133:](#2uxtw84)  \*/  
[134:](#1a346fx)  public function to\_array($data = NULL)  
[135:](#3u2rp3q)  {  
[136:](#2981zbj)  // If no data is passed as a parameter, then use the data passed  
[137:](#odc9jc)  // via the constructor  
[138:](#38czs75)  if ($data === NULL && func\_num\_args() === 0)  
[139:](#1nia2ey)  {  
[140:](#47hxl2r)  $data = $this->\_data;  
[141:](#2mn7vak)  }  
[142:](#11si5id)   
[143:](#3ls5o66)  // Cast as an array if not already  
[144:](#20xfydz)  if (is\_array($data) === FALSE)  
[145:](#4kx3h1s)  {  
[146:](#302dr9l)  $data = (array) $data;  
[147:](#1f7o1he)  }  
[148:](#3z7bk57)   
[149:](#2eclud0)  $array = [];  
[150:](#thw4kt)  foreach ((array) $data as $key => $value)  
[151:](#3dhjn8m)  {  
[152:](#1smtxgf)  if (is\_object($value) === TRUE || is\_array($value) === TRUE)  
[153:](#4cmhg48)  {  
[154:](#2rrrqc1)  $array[$key] = $this->to\_array($value);  
[155:](#16x20ju)  }  
[156:](#3qwpj7n)  else  
[157:](#261ztfg)  {  
[158:](#l7a3n9)  $array[$key] = $value;  
[159:](#356xmb2)  }  
[160:](#1kc7wiv)  }  
[161:](#44bvf6o)   
[162:](#2jh5peh)  return $array;  
[163:](#ymfzma)  }  
[164:](#3im3ia3)   
[165:](#1xrdshw)  /\*\*  
[166:](#4hr1b5p)  \* Format data as XML  
[167:](#2wwbldi)  \*  
[168:](#1c1lvlb)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[169:](#3w19e94)  \* to the constructor  
[170:](#2b6jogx)  \* @param NULL $structure  
[171:](#qbtyoq)  \* @param string $basenode  
[172:](#3abhhcj)  \* @return mixed  
[173:](#1pgrrkc)  \*/  
[174:](#49gfa85)  public function to\_xml($data = NULL, $structure = NULL, $basenode = 'xml')  
[175:](#2olpkfy)  {  
[176:](#13qzunr)  if ($data === NULL && func\_num\_args() === 0)  
[177:](#3nqndbk)  {  
[178:](#22vxnjd)  $data = $this->\_data;  
[179:](#i17xr6)  }  
[180:](#320vgez)   
[181:](#1h65qms)  // turn off compatibility mode as simple xml throws a wobbly if you don't.  
[182:](#415t9al)  if (ini\_get('zend.ze1\_compatibility\_mode') == 1)  
[183:](#2gb3jie)  {  
[184:](#vgdtq7)  ini\_set('zend.ze1\_compatibility\_mode', 0);  
[185:](#3fg1ce0)  }  
[186:](#1ulbmlt)   
[187:](#4ekz59m)  if ($structure === NULL)  
[188:](#2tq9fhf)  {  
[189:](#18vjpp8)  $structure = simplexml\_load\_string("<?xml version='1.0' encoding='utf-8'?><$basenode />");  
[190:](#3sv78d1)  }  
[191:](#280hiku)   
[192:](#n5rssn)  // Force it to be something useful  
[193:](#375fbgg)  if (is\_array($data) === FALSE && is\_object($data) === FALSE)  
[194:](#1maplo9)  {  
[195:](#46ad4c2)  $data = (array) $data;  
[196:](#2lfnejv)  }  
[197:](#10kxoro)   
[198:](#3kkl7fh)  foreach ($data as $key => $value)  
[199:](#1zpvhna)  {  
[200:](#4jpj0b3)   
[201:](#2yutaiw)  //change false/true to 0/1  
[202:](#1e03kqp)  if (is\_bool($value))  
[203:](#3xzr3ei)  {  
[204:](#2d51dmb)  $value = (int) $value;  
[205:](#sabnu4)  }  
[206:](#3c9z6hx)   
[207:](#1rf9gpq)  // no numeric keys in our xml please!  
[208:](#4bewzdj)  if (is\_numeric($key))  
[209:](#2qk79lc)  {  
[210:](#15phjt5)  // make string key...  
[211:](#3pp52gy)  $key = (singular($basenode) != $basenode) ? singular($basenode) : 'item';  
[212:](#24ufcor)  }  
[213:](#jzpmwk)   
[214:](#33zd5kd)  // replace anything not alpha numeric  
[215:](#1j4nfs6)  $key = preg\_replace('/[^a-z\_\-0-9]/i', '', $key);  
[216:](#434ayfz)   
[217:](#2i9l8ns)  if ($key === '\_attributes' && (is\_array($value) || is\_object($value)))  
[218:](#xevivl)  {  
[219:](#3hej1je)  $attributes = $value;  
[220:](#1wjtbr7)  if (is\_object($attributes))  
[221:](#4gjguf0)  {  
[222:](#2vor4mt)  $attributes = get\_object\_vars($attributes);  
[223:](#1au1eum)  }  
[224:](#3utoxif)   
[225:](#29yz7q8)  foreach ($attributes as $attribute\_name => $attribute\_value)  
[226:](#p49hy1)  {  
[227:](#393x0lu)  $structure->addAttribute($attribute\_name, $attribute\_value);  
[228:](#1o97atn)  }  
[229:](#488uthg)  }  
[230:](#2ne53p9)  // if there is another array found recursively call this function  
[231:](#12jfdx2)  elseif (is\_array($value) || is\_object($value))  
[232:](#3mj2wkv)  {  
[233:](#21od6so)  $node = $structure->addChild($key);  
[234:](#gtnh0h)   
[235:](#30tazoa)  // recursive call.  
[236:](#1fyl9w3)  $this->to\_xml($value, $node, $key);  
[237:](#3zy8sjw)  }  
[238:](#2f3j2rp)  else  
[239:](#u8tczi)  {  
[240:](#3e8gvnb)  // add single node.  
[241:](#1tdr5v4)  $value = htmlspecialchars(html\_entity\_decode($value, ENT\_QUOTES, 'UTF-8'), ENT\_QUOTES, 'UTF-8');  
[242:](#4ddeoix)   
[243:](#2sioyqq)  $structure->addChild($key, $value);  
[244:](#17nz8yj)  }  
[245:](#3rnmrmc)  }  
[246:](#26sx1u5)   
[247:](#ly7c1y)  return $structure->asXML();  
[248:](#35xuupr)  }  
[249:](#1l354xk)   
[250:](#452snld)  /\*\*  
[251:](#2k82xt6)  \* Format data as HTML  
[252:](#zdd80z)  \*  
[253:](#3jd0qos)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[254:](#1yib0wl)  \* to the constructor  
[255:](#4ihyjke)  \* @return mixed  
[256:](#2xn8ts7)  \*/  
[257:](#1csj400)  public function to\_html($data = NULL)  
[258:](#3ws6mnt)  {  
[259:](#2bxgwvm)  // If no data is passed as a parameter, then use the data passed  
[260:](#r2r73f)  // via the constructor  
[261:](#3b2epr8)  if ($data === NULL && func\_num\_args() === 0)  
[262:](#1q7ozz1)  {  
[263:](#4a7cimu)  $data = $this->\_data;  
[264:](#2pcmsun)  }  
[265:](#14hx32g)   
[266:](#3ohklq9)  // Cast as an array if not already  
[267:](#23muvy2)  if (is\_array($data) === FALSE)  
[268:](#is565v)  {  
[269:](#32rsoto)  $data = (array) $data;  
[270:](#1hx2z1h)  }  
[271:](#41wqhpa)   
[272:](#2h20rx3)  // Check if it's a multi-dimensional array  
[273:](#w7b24w)  if (isset($data[0]) && count($data) !== count($data, COUNT\_RECURSIVE))  
[274:](#3g6yksp)  {  
[275:](#1vc8v0i)  // Multi-dimensional array  
[276:](#4fbwdob)  $headings = array\_keys($data[0]);  
[277:](#2uh6nw4)  }  
[278:](#19mgy3x)  else  
[279:](#3tm4grq)  {  
[280:](#28reqzj)  // Single array  
[281:](#nwp17c)  $headings = array\_keys($data);  
[282:](#37wcjv5)  $data = [$data];  
[283:](#1n1mu2y)  }  
[284:](#471acqr)   
[285:](#2m6kmyk)  // Load the table library  
[286:](#11bux6d)  $this->\_CI->load->library('table');  
[287:](#3lbifu6)   
[288:](#20gsq1z)  $this->\_CI->table->set\_heading($headings);  
[289:](#4kgg8ps)   
[290:](#2zlqixl)  foreach ($data as $row)  
[291:](#1er0t5e)  {  
[292:](#3yqobt7)  // Suppressing the "array to string conversion" notice  
[293:](#2dvym10)  // Keep the "evil" @ here  
[294:](#t18w8t)  $row = @array\_map('strval', $row);  
[295:](#3d0wewm)   
[296:](#1s66p4f)  $this->\_CI->table->add\_row($row);  
[297:](#4c5u7s8)  }  
[298:](#2rb4i01)   
[299:](#16ges7u)  return $this->\_CI->table->generate();  
[300:](#3qg2avn)  }  
[301:](#25lcl3g)   
[302:](#kqmvb9)  /\*\*  
[303:](#34qadz2)  \* @link http://www.metashock.de/2014/02/create-csv-file-in-memory-php/  
[304:](#1jvko6v)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[305:](#43v86uo)  \* to the constructor  
[306:](#2j0ih2h)  \* @param string $delimiter The optional delimiter parameter sets the field  
[307:](#y5sraa)  \* delimiter (one character only). NULL will use the default value (,)  
[308:](#3i5g9y3)  \* @param string $enclosure The optional enclosure parameter sets the field  
[309:](#1xaqk5w)  \* enclosure (one character only). NULL will use the default value (")  
[310:](#4hae2tp)  \* @return string A csv string  
[311:](#2wfod1i)  \*/  
[312:](#1bkyn9b)  public function to\_csv($data = NULL, $delimiter = ',', $enclosure = '"')  
[313:](#3vkm5x4)  {  
[314:](#2apwg4x)  // Use a threshold of 1 MB (1024 \* 1024)  
[315:](#pv6qcq)  $handle = fopen('php://temp/maxmemory:1048576', 'w');  
[316:](#39uu90j)  if ($handle === FALSE)  
[317:](#1p04j8c)  {  
[318:](#48zs1w5)  return NULL;  
[319:](#2o52c3y)  }  
[320:](#13acmbr)   
[321:](#3na04zk)  // If no data is passed as a parameter, then use the data passed  
[322:](#22faf7d)  // via the constructor  
[323:](#hkkpf6)  if ($data === NULL && func\_num\_args() === 0)  
[324:](#31k882z)  {  
[325:](#1gpiias)  $data = $this->\_data;  
[326:](#40p60yl)  }  
[327:](#2fugb6e)   
[328:](#uzqle7)  // If NULL, then set as the default delimiter  
[329:](#3eze420)  if ($delimiter === NULL)  
[330:](#1u4oe9t)  {  
[331:](#4e4bwxm)  $delimiter = ',';  
[332:](#2t9m75f)  }  
[333:](#18ewhd8)   
[334:](#3sek011)  // If NULL, then set as the default enclosure  
[335:](#27jua8u)  if ($enclosure === NULL)  
[336:](#mp4kgn)  {  
[337:](#36os34g)  $enclosure = '"';  
[338:](#1lu2dc9)  }  
[339:](#45tpw02)   
[340:](#2kz067v)  // Cast as an array if not already  
[341:](#104agfo)  if (is\_array($data) === FALSE)  
[342:](#3k3xz3h)  {  
[343:](#1z989ba)  $data = (array) $data;  
[344:](#4j8vrz3)  }  
[345:](#2ye626w)   
[346:](#1djgcep)  // Check if it's a multi-dimensional array  
[347:](#3xj3v2i)  if (isset($data[0]) && count($data) !== count($data, COUNT\_RECURSIVE))  
[348:](#2coe5ab)  {  
[349:](#rtofi4)  // Multi-dimensional array  
[350:](#3btby5x)  $headings = array\_keys($data[0]);  
[351:](#1qym8dq)  }  
[352:](#4ay9r1j)  else  
[353:](#2q3k19c)  {  
[354:](#158ubh5)  // Single array  
[355:](#3p8hu4y)  $headings = array\_keys($data);  
[356:](#24ds4cr)  $data = [$data];  
[357:](#jj2ekk)  }  
[358:](#33ipx8d)   
[359:](#1io07g6)  // Apply the headings  
[360:](#42nnq3z)  fputcsv($handle, $headings, $delimiter, $enclosure);  
[361:](#2hsy0bs)   
[362:](#wy8ajl)  foreach ($data as $record)  
[363:](#3gxvt7e)  {  
[364:](#1w363f7)  // If the record is not an array, then break. This is because the 2nd param of  
[365:](#4g2tm30)  // fputcsv() should be an array  
[366:](#2v83wat)  if (is\_array($record) === FALSE)  
[367:](#1ade6im)  {  
[368:](#3ud1p6f)  break;  
[369:](#29ibze8)  }  
[370:](#onm9m1)   
[371:](#38n9s9u)  // Suppressing the "array to string conversion" notice.  
[372:](#1nsk2hn)  // Keep the "evil" @ here.  
[373:](#47s7l5g)  $record = @ array\_map('strval', $record);  
[374:](#2mxhvd9)   
[375:](#122s5l2)  // Returns the length of the string written or FALSE  
[376:](#3m2fo8v)  fputcsv($handle, $record, $delimiter, $enclosure);  
[377:](#217pygo)  }  
[378:](#4l7dh4h)   
[379:](#30cnrca)  // Reset the file pointer  
[380:](#1fhy1k3)  rewind($handle);  
[381:](#3zhlk7w)   
[382:](#2emvufp)  // Retrieve the csv contents  
[383:](#ts64ni)  $csv = stream\_get\_contents($handle);  
[384:](#3drtnbb)   
[385:](#1sx3xj4)  // Close the handle  
[386:](#4cwrg6x)  fclose($handle);  
[387:](#2s21qeq)   
[388:](#177c0mj)  return $csv;  
[389:](#3r6zjac)  }  
[390:](#26c9ti5)   
[391:](#lhk3py)  /\*\*  
[392:](#35h7mdr)  \* Encode data as json  
[393:](#1kmhwlk)  \*  
[394:](#44m5f9d)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[395:](#2jrfph6)  \* to the constructor  
[396:](#ywpzoz)  \* @return string Json representation of a value  
[397:](#3iwdics)  \*/  
[398:](#1y1nskl)  public function to\_json($data = NULL)  
[399:](#4i1bb8e)  {  
[400:](#2x6llg7)  // If no data is passed as a parameter, then use the data passed  
[401:](#1cbvvo0)  // via the constructor  
[402:](#3wbjebt)  if ($data === NULL && func\_num\_args() === 0)  
[403:](#2bgtojm)  {  
[404:](#qm3yrf)  $data = $this->\_data;  
[405:](#3alrhf8)  }  
[406:](#1pr1rn1)   
[407:](#49qpaau)  // Get the callback parameter (if set)  
[408:](#2ovzkin)  $callback = $this->\_CI->input->get('callback');  
[409:](#1419uqg)   
[410:](#3o0xde9)  if (empty($callback) === TRUE)  
[411:](#2367nm2)  {  
[412:](#ibhxtv)  return json\_encode($data);  
[413:](#32b5gho)  }  
[414:](#1hgfqph)   
[415:](#41g39da)  // We only honour a jsonp callback which are valid javascript identifiers  
[416:](#2gldjl3)  elseif (preg\_match('/^[a-z\_\$][a-z0-9\$\_]\*(\.[a-z\_\$][a-z0-9\$\_]\*)\*$/i', $callback))  
[417:](#vqntsw)  {  
[418:](#3fqbcgp)  // Return the data as encoded json with a callback  
[419:](#1uvlmoi)  return $callback.'('.json\_encode($data).');';  
[420:](#4ev95cb)  }  
[421:](#2u0jfk4)   
[422:](#195tprx)  // An invalid jsonp callback function provided.  
[423:](#3t5h8fq)  // Though I don't believe this should be hardcoded here  
[424:](#28arinj)  $data['warning'] = 'INVALID JSONP CALLBACK: '.$callback;  
[425:](#ng1svc)   
[426:](#37fpbj5)  return json\_encode($data);  
[427:](#1mkzlqy)  }  
[428:](#46kn4er)   
[429:](#2lpxemk)  /\*\*  
[430:](#10v7oud)  \* Encode data as a serialized array  
[431:](#3kuv7i6)  \*  
[432:](#2005hpz)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[433:](#4jzt0ds)  \* to the constructor  
[434:](#2z53all)  \* @return string Serialized data  
[435:](#1eadkte)  \*/  
[436:](#3ya13h7)  public function to\_serialized($data = NULL)  
[437:](#2dfbdp0)  {  
[438:](#sklnwt)  // If no data is passed as a parameter, then use the data passed  
[439:](#3ck96km)  // via the constructor  
[440:](#1rpjgsf)  if ($data === NULL && func\_num\_args() === 0)  
[441:](#4bp6zg8)  {  
[442:](#2quh9o1)  $data = $this->\_data;  
[443:](#15zrjvu)  }  
[444:](#3pzf2jn)   
[445:](#254pcrg)  return serialize($data);  
[446:](#k9zmz9)  }  
[447:](#349n5n2)   
[448:](#1jexfuv)  /\*\*  
[449:](#43ekyio)  \* Format data using a PHP structure  
[450:](#2ijv8qh)  \*  
[451:](#xp5iya)  \* @param mixed|NULL $data Optional data to pass, so as to override the data passed  
[452:](#3hot1m3)  \* to the constructor  
[453:](#1wu3btw)  \* @return mixed String representation of a variable  
[454:](#4gtquhp)  \*/  
[455:](#2vz14pi)  public function to\_php($data = NULL)  
[456:](#1b4bexb)  {  
[457:](#3v3yxl4)  // If no data is passed as a parameter, then use the data passed  
[458:](#2a997sx)  // via the constructor  
[459:](#peji0q)  if ($data === NULL && func\_num\_args() === 0)  
[460:](#39e70oj)  {  
[461:](#1ojhawc)  $data = $this->\_data;  
[462:](#48j4tk5)  }  
[463:](#2nof3ry)   
[464:](#12tpdzr)  return var\_export($data, TRUE);  
[465:](#3mtcwnk)  }  
[466:](#21yn6vd)   
[467:](#h3xh36)  // INTERNAL FUNCTIONS  
[468:](#313kzqz)   
[469:](#1g8v9ys)  /\*\*  
[470:](#408isml)  \* @param $data XML string  
[471:](#2fdt2ue)  \* @return SimpleXMLElement XML element object; otherwise, empty array  
[472:](#uj3d27)  \*/  
[473:](#3eiqvq0)  protected function \_from\_xml($data)  
[474:](#1to15xt)  {  
[475:](#4dnoolm)  return $data ? (array) simplexml\_load\_string($data, 'SimpleXMLElement', LIBXML\_NOCDATA) : [];  
[476:](#2ssyytf)  }  
[477:](#17y9918)   
[478:](#3rxwrp1)  /\*\*  
[479:](#27371wu)  \* @param string $data CSV string  
[480:](#m8hc4n)  \* @param string $delimiter The optional delimiter parameter sets the field  
[481:](#3684usg)  \* delimiter (one character only). NULL will use the default value (,)  
[482:](#1ldf509)  \* @param string $enclosure The optional enclosure parameter sets the field  
[483:](#45d2no2)  \* enclosure (one character only). NULL will use the default value (")  
[484:](#2kicxvv)  \* @return array A multi-dimensional array with the outer array being the number of rows  
[485:](#znn83o)  \* and the inner arrays the individual fields  
[486:](#3jnaqrh)  \*/  
[487:](#1ysl0za)  protected function \_from\_csv($data, $delimiter = ',', $enclosure = '"')  
[488:](#4is8jn3)  {  
[489:](#2xxituw)  // If NULL, then set as the default delimiter  
[490:](#1d2t42p)  if ($delimiter === NULL)  
[491:](#3x2gmqi)  {  
[492:](#2c7qwyb)  $delimiter = ',';  
[493:](#rd1764)  }  
[494:](#3bcoptx)   
[495:](#1qhz01q)  // If NULL, then set as the default enclosure  
[496:](#4ahmipj)  if ($enclosure === NULL)  
[497:](#2pmwsxc)  {  
[498:](#14s7355)  $enclosure = '"';  
[499:](#3orulsy)  }  
[500:](#23x4w0r)   
[501:](#j2f68k)  return str\_getcsv($data, $delimiter, $enclosure);  
[502:](#3322owd)  }  
[503:](#1i7cz46)   
[504:](#4270hrz)  /\*\*  
[505:](#2hcarzs)  \* @param $data Encoded json string  
[506:](#whl27l)  \* @return mixed Decoded json string with leading and trailing whitespace removed  
[507:](#3gh8kve)  \*/  
[508:](#1vmiv37)  protected function \_from\_json($data)  
[509:](#4fm6dr0)  {  
[510:](#2urgnyt)  return json\_decode(trim($data));  
[511:](#19wqy6m)  }  
[512:](#3tweguf)   
[513:](#291or28)  /\*\*  
[514:](#o6z1a1)  \* @param string Data to unserialized  
[515:](#386mjxu)  \* @return mixed Unserialized data  
[516:](#1nbwu5n)  \*/  
[517:](#47bkctg)  protected function \_from\_serialize($data)  
[518:](#2mgun19)  {  
[519:](#11m4x92)  return unserialize(trim($data));  
[520:](#3llsfwv)  }  
[521:](#20r2q4o)   
[522:](#4kqq8sh)  /\*\*  
[523:](#2zw0j0a)  \* @param $data Data to trim leading and trailing whitespace  
[524:](#1f1at83)  \* @return string Data with leading and trailing whitespace removed  
[525:](#3z0ybvw)  \*/  
[526:](#2e68m3p)  protected function \_from\_php($data)  
[527:](#tbiwbi)  {  
[528:](#3db6ezb)  return trim($data);  
[529:](#1sggp74)  }  
[530:](#4cg47ux)   
[531:](#2rlei2q) }  
[532:](#16qosaj)

API documentation generated by [ApiGen](http://apigen.org)