[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)  \* CodeIgniter Rest Controller  
 [7:](#3dy6vkm)  \* A fully RESTful server implementation for CodeIgniter using one library, one config file and one controller.  
 [8:](#1t3h5sf)  \*  
 [9:](#4d34og8)  \* @package CodeIgniter  
 [10:](#2s8eyo1)  \* @subpackage Libraries  
 [11:](#17dp8vu)  \* @category Libraries  
 [12:](#3rdcrjn)  \* @author Phil Sturgeon, Chris Kacerguis  
 [13:](#26in1rg)  \* @license MIT  
 [14:](#lnxbz9)  \* @link https://github.com/chriskacerguis/codeigniter-restserver  
 [15:](#35nkun2)  \* @version 3.0.0  
 [16:](#1ksv4uv)  \*/  
 [17:](#44sinio) abstract class REST\_Controller extends CI\_Controller {  
 [18:](#2jxsxqh)   
 [19:](#z337ya)  // Note: Only the widely used HTTP status codes are documented  
 [20:](#3j2qqm3)   
 [21:](#1y810tw)  // Informational  
 [22:](#4i7ojhp)   
 [23:](#2xcytpi)  const HTTP\_CONTINUE = 100;  
 [24:](#1ci93xb)  const HTTP\_SWITCHING\_PROTOCOLS = 101;  
 [25:](#3whwml4)  const HTTP\_PROCESSING = 102; // RFC2518  
 [26:](#2bn6wsx)   
 [27:](#qsh70q)  // Success  
 [28:](#3as4poj)   
 [29:](#1pxezwc)  /\*\*  
 [30:](#49x2ik5)  \* The request has succeeded  
 [31:](#2p2csry)  \*/  
 [32:](#147n2zr)  const HTTP\_OK = 200;  
 [33:](#3o7alnk)   
 [34:](#23ckvvd)  /\*\*  
 [35:](#ihv636)  \* The server successfully created a new resource  
 [36:](#32hioqz)  \*/  
 [37:](#1hmsyys)  const HTTP\_CREATED = 201;  
 [38:](#41mghml)  const HTTP\_ACCEPTED = 202;  
 [39:](#2grqrue)  const HTTP\_NON\_AUTHORITATIVE\_INFORMATION = 203;  
 [40:](#vx1227)   
 [41:](#3fwokq0)  /\*\*  
 [42:](#1v1yuxt)  \* The server successfully processed the request, though no content is returned  
 [43:](#4f1mdlm)  \*/  
 [44:](#2u6wntf)  const HTTP\_NO\_CONTENT = 204;  
 [45:](#19c6y18)  const HTTP\_RESET\_CONTENT = 205;  
 [46:](#3tbugp1)  const HTTP\_PARTIAL\_CONTENT = 206;  
 [47:](#28h4qwu)  const HTTP\_MULTI\_STATUS = 207; // RFC4918  
 [48:](#nmf14n)  const HTTP\_ALREADY\_REPORTED = 208; // RFC5842  
 [49:](#37m2jsg)  const HTTP\_IM\_USED = 226; // RFC3229  
 [50:](#1mrcu09)   
 [51:](#46r0co2)  // Redirection  
 [52:](#2lwamvv)   
 [53:](#111kx3o)  const HTTP\_MULTIPLE\_CHOICES = 300;  
 [54:](#3l18frh)  const HTTP\_MOVED\_PERMANENTLY = 301;  
 [55:](#206ipza)  const HTTP\_FOUND = 302;  
 [56:](#4k668n3)  const HTTP\_SEE\_OTHER = 303;  
 [57:](#2zbgiuw)   
 [58:](#1egqt2p)  /\*\*  
 [59:](#3ygebqi)  \* The resource has not been modified since the last request  
 [60:](#2dlolyb)  \*/  
 [61:](#sqyw64)  const HTTP\_NOT\_MODIFIED = 304;  
 [62:](#3cqmetx)  const HTTP\_USE\_PROXY = 305;  
 [63:](#1rvwp1q)  const HTTP\_RESERVED = 306;  
 [64:](#4bvk7pj)  const HTTP\_TEMPORARY\_REDIRECT = 307;  
 [65:](#2r0uhxc)  const HTTP\_PERMANENTLY\_REDIRECT = 308; // RFC7238  
 [66:](#1664s55)   
 [67:](#3q5sasy)  // Client Error  
 [68:](#25b2l0r)   
 [69:](#kgcv8k)  /\*\*  
 [70:](#34g0dwd)  \* The request cannot be fulfilled due to multiple errors  
 [71:](#1jlao46)  \*/  
 [72:](#43ky6rz)  const HTTP\_BAD\_REQUEST = 400;  
 [73:](#2iq8gzs)   
 [74:](#xvir7l)  /\*\*  
 [75:](#3hv69ve)  \* The user is unauthorized to access the requested resource  
 [76:](#1x0gk37)  \*/  
 [77:](#4h042r0)  const HTTP\_UNAUTHORIZED = 401;  
 [78:](#2w5ecyt)  const HTTP\_PAYMENT\_REQUIRED = 402;  
 [79:](#1baon6m)   
 [80:](#3vac5uf)  /\*\*  
 [81:](#2afmg28)  \* The requested resource is unavailable at this present time  
 [82:](#pkwqa1)  \*/  
 [83:](#39kk8xu)  const HTTP\_FORBIDDEN = 403;  
 [84:](#1opuj5n)   
 [85:](#48pi1tg)  /\*\*  
 [86:](#2nusc19)  \* The requested resource could not be found  
 [87:](#1302m92)  \*  
 [88:](#3mzq4wv)  \* Note: This is sometimes used to mask if there was an UNAUTHORIZED (401) or  
 [89:](#2250f4o)  \* FORBIDDEN (403) error, for security reasons  
 [90:](#haapch)  \*/  
 [91:](#319y80a)  const HTTP\_NOT\_FOUND = 404;  
 [92:](#1gf8i83)   
 [93:](#40ew0vw)  /\*\*  
 [94:](#2fk6b3p)  \* The request method is not supported by the following resource  
 [95:](#upglbi)  \*/  
 [96:](#3ep43zb)  const HTTP\_METHOD\_NOT\_ALLOWED = 405;  
 [97:](#1tuee74)   
 [98:](#4du1wux)  /\*\*  
 [99:](#2szc72q)  \* The request was not acceptable  
 [100:](#184mhaj)  \*/  
 [101:](#3s49zyc)  const HTTP\_NOT\_ACCEPTABLE = 406;  
 [102:](#279ka65)  const HTTP\_PROXY\_AUTHENTICATION\_REQUIRED = 407;  
 [103:](#meukdy)  const HTTP\_REQUEST\_TIMEOUT = 408;  
 [104:](#36ei31r)   
 [105:](#1ljsd9k)  /\*\*  
 [106:](#45jfvxd)  \* The request could not be completed due to a conflict with the current state  
 [107:](#2koq656)  \* of the resource  
 [108:](#zu0gcz)  \*/  
 [109:](#3jtnz0s)  const HTTP\_CONFLICT = 409;  
 [110:](#1yyy98l)  const HTTP\_GONE = 410;  
 [111:](#4iylrwe)  const HTTP\_LENGTH\_REQUIRED = 411;  
 [112:](#2y3w247)  const HTTP\_PRECONDITION\_FAILED = 412;  
 [113:](#1d96cc0)  const HTTP\_REQUEST\_ENTITY\_TOO\_LARGE = 413;  
 [114:](#3x8tuzt)  const HTTP\_REQUEST\_URI\_TOO\_LONG = 414;  
 [115:](#2ce457m)  const HTTP\_UNSUPPORTED\_MEDIA\_TYPE = 415;  
 [116:](#rjefff)  const HTTP\_REQUESTED\_RANGE\_NOT\_SATISFIABLE = 416;  
 [117:](#3bj1y38)  const HTTP\_EXPECTATION\_FAILED = 417;  
 [118:](#1qoc8b1)  const HTTP\_I\_AM\_A\_TEAPOT = 418; // RFC2324  
 [119:](#4anzqyu)  const HTTP\_UNPROCESSABLE\_ENTITY = 422; // RFC4918  
 [120:](#2pta16n)  const HTTP\_LOCKED = 423; // RFC4918  
 [121:](#14ykbeg)  const HTTP\_FAILED\_DEPENDENCY = 424; // RFC4918  
 [122:](#3oy7u29)  const HTTP\_RESERVED\_FOR\_WEBDAV\_ADVANCED\_COLLECTIONS\_EXPIRED\_PROPOSAL = 425; // RFC2817  
 [123:](#243i4a2)  const HTTP\_UPGRADE\_REQUIRED = 426; // RFC2817  
 [124:](#j8sehv)  const HTTP\_PRECONDITION\_REQUIRED = 428; // RFC6585  
 [125:](#338fx5o)  const HTTP\_TOO\_MANY\_REQUESTS = 429; // RFC6585  
 [126:](#1idq7dh)  const HTTP\_REQUEST\_HEADER\_FIELDS\_TOO\_LARGE = 431; // RFC6585  
 [127:](#42ddq1a)   
 [128:](#2hio093)  // Server Error  
 [129:](#wnyagw)   
 [130:](#3gnlt4p)  /\*\*  
 [131:](#1vsw3ci)  \* The server encountered an unexpected error  
 [132:](#4fsjm0b)  \*  
 [133:](#2uxtw84)  \* Note: This is a generic error message when no specific message  
 [134:](#1a346fx)  \* is suitable  
 [135:](#3u2rp3q)  \*/  
 [136:](#2981zbj)  const HTTP\_INTERNAL\_SERVER\_ERROR = 500;  
 [137:](#odc9jc)   
 [138:](#38czs75)  /\*\*  
 [139:](#1nia2ey)  \* The server does not recognise the request method  
 [140:](#47hxl2r)  \*/  
 [141:](#2mn7vak)  const HTTP\_NOT\_IMPLEMENTED = 501;  
 [142:](#11si5id)  const HTTP\_BAD\_GATEWAY = 502;  
 [143:](#3ls5o66)  const HTTP\_SERVICE\_UNAVAILABLE = 503;  
 [144:](#20xfydz)  const HTTP\_GATEWAY\_TIMEOUT = 504;  
 [145:](#4kx3h1s)  const HTTP\_VERSION\_NOT\_SUPPORTED = 505;  
 [146:](#302dr9l)  const HTTP\_VARIANT\_ALSO\_NEGOTIATES\_EXPERIMENTAL = 506; // RFC2295  
 [147:](#1f7o1he)  const HTTP\_INSUFFICIENT\_STORAGE = 507; // RFC4918  
 [148:](#3z7bk57)  const HTTP\_LOOP\_DETECTED = 508; // RFC5842  
 [149:](#2eclud0)  const HTTP\_NOT\_EXTENDED = 510; // RFC2774  
 [150:](#thw4kt)  const HTTP\_NETWORK\_AUTHENTICATION\_REQUIRED = 511;  
 [151:](#3dhjn8m)   
 [152:](#1smtxgf)  /\*\*  
 [153:](#4cmhg48)  \* This defines the rest format  
 [154:](#2rrrqc1)  \* Must be overridden it in a controller so that it is set  
 [155:](#16x20ju)  \*  
 [156:](#3qwpj7n)  \* @var string|NULL  
 [157:](#261ztfg)  \*/  
 [158:](#l7a3n9)  protected $rest\_format = NULL;  
 [159:](#356xmb2)   
 [160:](#1kc7wiv)  /\*\*  
 [161:](#44bvf6o)  \* Defines the list of method properties such as limit, log and level  
 [162:](#2jh5peh)  \*  
 [163:](#ymfzma)  \* @var array  
 [164:](#3im3ia3)  \*/  
 [165:](#1xrdshw)  protected $methods = [];  
 [166:](#4hr1b5p)   
 [167:](#2wwbldi)  /\*\*  
 [168:](#1c1lvlb)  \* List of allowed HTTP methods  
 [169:](#3w19e94)  \*  
 [170:](#2b6jogx)  \* @var array  
 [171:](#qbtyoq)  \*/  
 [172:](#3abhhcj)  protected $allowed\_http\_methods = ['get', 'delete', 'post', 'put', 'options', 'patch', 'head'];  
 [173:](#1pgrrkc)   
 [174:](#49gfa85)  /\*\*  
 [175:](#2olpkfy)  \* Contains details about the request  
 [176:](#13qzunr)  \* Fields: body, format, method, ssl  
 [177:](#3nqndbk)  \* Note: This is a dynamic object (stdClass)  
 [178:](#22vxnjd)  \*  
 [179:](#i17xr6)  \* @var object  
 [180:](#320vgez)  \*/  
 [181:](#1h65qms)  protected $request = NULL;  
 [182:](#415t9al)   
 [183:](#2gb3jie)  /\*\*  
 [184:](#vgdtq7)  \* Contains details about the response  
 [185:](#3fg1ce0)  \* Fields: format, lang  
 [186:](#1ulbmlt)  \* Note: This is a dynamic object (stdClass)  
 [187:](#4ekz59m)  \*  
 [188:](#2tq9fhf)  \* @var object  
 [189:](#18vjpp8)  \*/  
 [190:](#3sv78d1)  protected $response = NULL;  
 [191:](#280hiku)   
 [192:](#n5rssn)  /\*\*  
 [193:](#375fbgg)  \* Contains details about the REST API  
 [194:](#1maplo9)  \* Fields: db, ignore\_limits, key, level, user\_id  
 [195:](#46ad4c2)  \* Note: This is a dynamic object (stdClass)  
 [196:](#2lfnejv)  \*  
 [197:](#10kxoro)  \* @var object  
 [198:](#3kkl7fh)  \*/  
 [199:](#1zpvhna)  protected $rest = NULL;  
 [200:](#4jpj0b3)   
 [201:](#2yutaiw)  /\*\*  
 [202:](#1e03kqp)  \* The arguments for the GET request method  
 [203:](#3xzr3ei)  \*  
 [204:](#2d51dmb)  \* @var array  
 [205:](#sabnu4)  \*/  
 [206:](#3c9z6hx)  protected $\_get\_args = [];  
 [207:](#1rf9gpq)   
 [208:](#4bewzdj)  /\*\*  
 [209:](#2qk79lc)  \* The arguments for the POST request method  
 [210:](#15phjt5)  \*  
 [211:](#3pp52gy)  \* @var array  
 [212:](#24ufcor)  \*/  
 [213:](#jzpmwk)  protected $\_post\_args = [];  
 [214:](#33zd5kd)   
 [215:](#1j4nfs6)  /\*\*  
 [216:](#434ayfz)  \* The arguments for the PUT request method  
 [217:](#2i9l8ns)  \*  
 [218:](#xevivl)  \* @var array  
 [219:](#3hej1je)  \*/  
 [220:](#1wjtbr7)  protected $\_put\_args = [];  
 [221:](#4gjguf0)   
 [222:](#2vor4mt)  /\*\*  
 [223:](#1au1eum)  \* The arguments for the DELETE request method  
 [224:](#3utoxif)  \*  
 [225:](#29yz7q8)  \* @var array  
 [226:](#p49hy1)  \*/  
 [227:](#393x0lu)  protected $\_delete\_args = [];  
 [228:](#1o97atn)   
 [229:](#488uthg)  /\*\*  
 [230:](#2ne53p9)  \* The arguments for the PATCH request method  
 [231:](#12jfdx2)  \*  
 [232:](#3mj2wkv)  \* @var array  
 [233:](#21od6so)  \*/  
 [234:](#gtnh0h)  protected $\_patch\_args = [];  
 [235:](#30tazoa)   
 [236:](#1fyl9w3)  /\*\*  
 [237:](#3zy8sjw)  \* The arguments for the HEAD request method  
 [238:](#2f3j2rp)  \*  
 [239:](#u8tczi)  \* @var array  
 [240:](#3e8gvnb)  \*/  
 [241:](#1tdr5v4)  protected $\_head\_args = [];  
 [242:](#4ddeoix)   
 [243:](#2sioyqq)  /\*\*  
 [244:](#17nz8yj)  \* The arguments for the OPTIONS request method  
 [245:](#3rnmrmc)  \*  
 [246:](#26sx1u5)  \* @var array  
 [247:](#ly7c1y)  \*/  
 [248:](#35xuupr)  protected $\_options\_args = [];  
 [249:](#1l354xk)   
 [250:](#452snld)  /\*\*  
 [251:](#2k82xt6)  \* The arguments for the query parameters  
 [252:](#zdd80z)  \*  
 [253:](#3jd0qos)  \* @var array  
 [254:](#1yib0wl)  \*/  
 [255:](#4ihyjke)  protected $\_query\_args = [];  
 [256:](#2xn8ts7)   
 [257:](#1csj400)  /\*\*  
 [258:](#3ws6mnt)  \* The arguments from GET, POST, PUT, DELETE, PATCH, HEAD and OPTIONS request methods combined  
 [259:](#2bxgwvm)  \*  
 [260:](#r2r73f)  \* @var array  
 [261:](#3b2epr8)  \*/  
 [262:](#1q7ozz1)  protected $\_args = [];  
 [263:](#4a7cimu)   
 [264:](#2pcmsun)  /\*\*  
 [265:](#14hx32g)  \* The insert\_id of the log entry (if we have one)  
 [266:](#3ohklq9)  \*  
 [267:](#23muvy2)  \* @var string  
 [268:](#is565v)  \*/  
 [269:](#32rsoto)  protected $\_insert\_id = '';  
 [270:](#1hx2z1h)   
 [271:](#41wqhpa)  /\*\*  
 [272:](#2h20rx3)  \* If the request is allowed based on the API key provided  
 [273:](#w7b24w)  \*  
 [274:](#3g6yksp)  \* @var bool  
 [275:](#1vc8v0i)  \*/  
 [276:](#4fbwdob)  protected $\_allow = TRUE;  
 [277:](#2uh6nw4)   
 [278:](#19mgy3x)  /\*\*  
 [279:](#3tm4grq)  \* The LDAP Distinguished Name of the User post authentication  
 [280:](#28reqzj)  \*  
 [281:](#nwp17c)  \* @var string  
 [282:](#37wcjv5)  \*/  
 [283:](#1n1mu2y)  protected $\_user\_ldap\_dn = '';  
 [284:](#471acqr)   
 [285:](#2m6kmyk)  /\*\*  
 [286:](#11bux6d)  \* The start of the response time from the server  
 [287:](#3lbifu6)  \*  
 [288:](#20gsq1z)  \* @var string  
 [289:](#4kgg8ps)  \*/  
 [290:](#2zlqixl)  protected $\_start\_rtime = '';  
 [291:](#1er0t5e)   
 [292:](#3yqobt7)  /\*\*  
 [293:](#2dvym10)  \* The end of the response time from the server  
 [294:](#t18w8t)  \*  
 [295:](#3d0wewm)  \* @var string  
 [296:](#1s66p4f)  \*/  
 [297:](#4c5u7s8)  protected $\_end\_rtime = '';  
 [298:](#2rb4i01)   
 [299:](#16ges7u)  /\*\*  
 [300:](#3qg2avn)  \* List all supported methods, the first will be the default format  
 [301:](#25lcl3g)  \*  
 [302:](#kqmvb9)  \* @var array  
 [303:](#34qadz2)  \*/  
 [304:](#1jvko6v)  protected $\_supported\_formats = [  
 [305:](#43v86uo)  'json' => 'application/json',  
 [306:](#2j0ih2h)  'array' => 'application/json',  
 [307:](#y5sraa)  'csv' => 'application/csv',  
 [308:](#3i5g9y3)  'html' => 'text/html',  
 [309:](#1xaqk5w)  'jsonp' => 'application/javascript',  
 [310:](#4hae2tp)  'php' => 'text/plain',  
 [311:](#2wfod1i)  'serialized' => 'application/vnd.php.serialized',  
 [312:](#1bkyn9b)  'xml' => 'application/xml'  
 [313:](#3vkm5x4)  ];  
 [314:](#2apwg4x)   
 [315:](#pv6qcq)  /\*\*  
 [316:](#39uu90j)  \* Information about the current API user  
 [317:](#1p04j8c)  \*  
 [318:](#48zs1w5)  \* @var object  
 [319:](#2o52c3y)  \*/  
 [320:](#13acmbr)  protected $\_apiuser;  
 [321:](#3na04zk)   
 [322:](#22faf7d)  /\*\*  
 [323:](#hkkpf6)  \* Whether or not to perform a CORS check and apply CORS headers to the request  
 [324:](#31k882z)  \*  
 [325:](#1gpiias)  \* @var bool  
 [326:](#40p60yl)  \*/  
 [327:](#2fugb6e)  protected $check\_cors = NULL;  
 [328:](#uzqle7)   
 [329:](#3eze420)  /\*\*  
 [330:](#1u4oe9t)  \* Enable XSS flag  
 [331:](#4e4bwxm)  \* Determines whether the XSS filter is always active when  
 [332:](#2t9m75f)  \* GET, OPTIONS, HEAD, POST, PUT, DELETE and PATCH data is encountered  
 [333:](#18ewhd8)  \* Set automatically based on config setting  
 [334:](#3sek011)  \*  
 [335:](#27jua8u)  \* @var bool  
 [336:](#mp4kgn)  \*/  
 [337:](#36os34g)  protected $\_enable\_xss = FALSE;  
 [338:](#1lu2dc9)   
 [339:](#45tpw02)  /\*\*  
 [340:](#2kz067v)  \* HTTP status codes and their respective description  
 [341:](#104agfo)  \* Note: Only the widely used HTTP status codes are used  
 [342:](#3k3xz3h)  \*  
 [343:](#1z989ba)  \* @var array  
 [344:](#4j8vrz3)  \* @link http://www.restapitutorial.com/httpstatuscodes.html  
 [345:](#2ye626w)  \*/  
 [346:](#1djgcep)  protected $http\_status\_codes = [  
 [347:](#3xj3v2i)  self::HTTP\_OK => 'OK',  
 [348:](#2coe5ab)  self::HTTP\_CREATED => 'CREATED',  
 [349:](#rtofi4)  self::HTTP\_NO\_CONTENT => 'NO CONTENT',  
 [350:](#3btby5x)  self::HTTP\_NOT\_MODIFIED => 'NOT MODIFIED',  
 [351:](#1qym8dq)  self::HTTP\_BAD\_REQUEST => 'BAD REQUEST',  
 [352:](#4ay9r1j)  self::HTTP\_UNAUTHORIZED => 'UNAUTHORIZED',  
 [353:](#2q3k19c)  self::HTTP\_FORBIDDEN => 'FORBIDDEN',  
 [354:](#158ubh5)  self::HTTP\_NOT\_FOUND => 'NOT FOUND',  
 [355:](#3p8hu4y)  self::HTTP\_METHOD\_NOT\_ALLOWED => 'METHOD NOT ALLOWED',  
 [356:](#24ds4cr)  self::HTTP\_NOT\_ACCEPTABLE => 'NOT ACCEPTABLE',  
 [357:](#jj2ekk)  self::HTTP\_CONFLICT => 'CONFLICT',  
 [358:](#33ipx8d)  self::HTTP\_INTERNAL\_SERVER\_ERROR => 'INTERNAL SERVER ERROR',  
 [359:](#1io07g6)  self::HTTP\_NOT\_IMPLEMENTED => 'NOT IMPLEMENTED'  
 [360:](#42nnq3z)  ];  
 [361:](#2hsy0bs)   
 [362:](#wy8ajl)  /\*\*  
 [363:](#3gxvt7e)  \* Extend this function to apply additional checking early on in the process  
 [364:](#1w363f7)  \*  
 [365:](#4g2tm30)  \* @access protected  
 [366:](#2v83wat)  \* @return void  
 [367:](#1ade6im)  \*/  
 [368:](#3ud1p6f)  protected function early\_checks()  
 [369:](#29ibze8)  {  
 [370:](#onm9m1)  }  
 [371:](#38n9s9u)   
 [372:](#1nsk2hn)  /\*\*  
 [373:](#47s7l5g)  \* Constructor for the REST API  
 [374:](#2mxhvd9)  \*  
 [375:](#122s5l2)  \* @access public  
 [376:](#3m2fo8v)  \* @param string $config Configuration filename minus the file extension  
 [377:](#217pygo)  \* e.g: my\_rest.php is passed as 'my\_rest'  
 [378:](#4l7dh4h)  \* @return void  
 [379:](#30cnrca)  \*/  
 [380:](#1fhy1k3)  public function \_\_construct($config = 'rest')  
 [381:](#3zhlk7w)  {  
 [382:](#2emvufp)  parent::\_\_construct();  
 [383:](#ts64ni)   
 [384:](#3drtnbb)  // Disable XML Entity (security vulnerability)  
 [385:](#1sx3xj4)  libxml\_disable\_entity\_loader(TRUE);  
 [386:](#4cwrg6x)   
 [387:](#2s21qeq)  // Check to see if PHP is equal to or greater than 5.4.x  
 [388:](#177c0mj)  if (is\_php('5.4') === FALSE)  
 [389:](#3r6zjac)  {  
 [390:](#26c9ti5)  // CodeIgniter 3 is recommended for v5.4 or above  
 [391:](#lhk3py)  throw new Exception('Using PHP v'.PHP\_VERSION.', though PHP v5.4 or greater is required');  
 [392:](#35h7mdr)  }  
 [393:](#1kmhwlk)   
 [394:](#44m5f9d)  // Check to see if this is CI 3.x  
 [395:](#2jrfph6)  if (explode('.', CI\_VERSION, 2)[0] < 3)  
 [396:](#ywpzoz)  {  
 [397:](#3iwdics)  throw new Exception('REST Server requires CodeIgniter 3.x');  
 [398:](#1y1nskl)  }  
 [399:](#4i1bb8e)   
 [400:](#2x6llg7)  // Set the default value of global xss filtering. Same approach as CodeIgniter 3  
 [401:](#1cbvvo0)  $this->\_enable\_xss = ($this->config->item('global\_xss\_filtering') === TRUE);  
 [402:](#3wbjebt)   
 [403:](#2bgtojm)  // Don't try to parse template variables like {elapsed\_time} and {memory\_usage}  
 [404:](#qm3yrf)  // when output is displayed for not damaging data accidentally  
 [405:](#3alrhf8)  $this->output->parse\_exec\_vars = FALSE;  
 [406:](#1pr1rn1)   
 [407:](#49qpaau)  // Start the timer for how long the request takes  
 [408:](#2ovzkin)  $this->\_start\_rtime = microtime(TRUE);  
 [409:](#1419uqg)   
 [410:](#3o0xde9)  // Load the rest.php configuration file  
 [411:](#2367nm2)  $this->load->config($config);  
 [412:](#ibhxtv)   
 [413:](#32b5gho)  // At present the library is bundled with REST\_Controller 2.5+, but will eventually be part of CodeIgniter (no citation)  
 [414:](#1hgfqph)  $this->load->library('format');  
 [415:](#41g39da)   
 [416:](#2gldjl3)  // Determine supported output formats from configuration  
 [417:](#vqntsw)  $supported\_formats = $this->config->item('rest\_supported\_formats');  
 [418:](#3fqbcgp)   
 [419:](#1uvlmoi)  // Validate the configuration setting output formats  
 [420:](#4ev95cb)  if (empty($supported\_formats))  
 [421:](#2u0jfk4)  {  
 [422:](#195tprx)  $supported\_formats = [];  
 [423:](#3t5h8fq)  }  
 [424:](#28arinj)   
 [425:](#ng1svc)  if ( ! is\_array($supported\_formats))  
 [426:](#37fpbj5)  {  
 [427:](#1mkzlqy)  $supported\_formats = [$supported\_formats];  
 [428:](#46kn4er)  }  
 [429:](#2lpxemk)   
 [430:](#10v7oud)  // Add silently the default output format if it is missing  
 [431:](#3kuv7i6)  $default\_format = $this->\_get\_default\_output\_format();  
 [432:](#2005hpz)  if (!in\_array($default\_format, $supported\_formats))  
 [433:](#4jzt0ds)  {  
 [434:](#2z53all)  $supported\_formats[] = $default\_format;  
 [435:](#1eadkte)  }  
 [436:](#3ya13h7)   
 [437:](#2dfbdp0)  // Now update $this->\_supported\_formats  
 [438:](#sklnwt)  $this->\_supported\_formats = array\_intersect\_key($this->\_supported\_formats, array\_flip($supported\_formats));  
 [439:](#3ck96km)   
 [440:](#1rpjgsf)  // Get the language  
 [441:](#4bp6zg8)  $language = $this->config->item('rest\_language');  
 [442:](#2quh9o1)  if ($language === NULL)  
 [443:](#15zrjvu)  {  
 [444:](#3pzf2jn)  $language = 'english';  
 [445:](#254pcrg)  }  
 [446:](#k9zmz9)   
 [447:](#349n5n2)  // Load the language file  
 [448:](#1jexfuv)  $this->lang->load('rest\_controller', $language);  
 [449:](#43ekyio)   
 [450:](#2ijv8qh)  // Initialise the response, request and rest objects  
 [451:](#xp5iya)  $this->request = new stdClass();  
 [452:](#3hot1m3)  $this->response = new stdClass();  
 [453:](#1wu3btw)  $this->rest = new stdClass();  
 [454:](#4gtquhp)   
 [455:](#2vz14pi)  // Check to see if the current IP address is blacklisted  
 [456:](#1b4bexb)  if ($this->config->item('rest\_ip\_blacklist\_enabled') === TRUE)  
 [457:](#3v3yxl4)  {  
 [458:](#2a997sx)  $this->\_check\_blacklist\_auth();  
 [459:](#peji0q)  }  
 [460:](#39e70oj)   
 [461:](#1ojhawc)  // Determine whether the connection is HTTPS  
 [462:](#48j4tk5)  $this->request->ssl = is\_https();  
 [463:](#2nof3ry)   
 [464:](#12tpdzr)  // How is this request being made? GET, POST, PATCH, DELETE, INSERT, PUT, HEAD or OPTIONS  
 [465:](#3mtcwnk)  $this->request->method = $this->\_detect\_method();  
 [466:](#21yn6vd)   
 [467:](#h3xh36)  // Check for CORS access request  
 [468:](#313kzqz)  $check\_cors = $this->config->item('check\_cors');  
 [469:](#1g8v9ys)  if ($check\_cors === TRUE)  
 [470:](#408isml)  {  
 [471:](#2fdt2ue)  $this->\_check\_cors();  
 [472:](#uj3d27)  }  
 [473:](#3eiqvq0)   
 [474:](#1to15xt)  // Create an argument container if it doesn't exist e.g. \_get\_args  
 [475:](#4dnoolm)  if (isset($this->{'\_'.$this->request->method.'\_args'}) === FALSE)  
 [476:](#2ssyytf)  {  
 [477:](#17y9918)  $this->{'\_'.$this->request->method.'\_args'} = [];  
 [478:](#3rxwrp1)  }  
 [479:](#27371wu)   
 [480:](#m8hc4n)  // Set up the query parameters  
 [481:](#3684usg)  $this->\_parse\_query();  
 [482:](#1ldf509)   
 [483:](#45d2no2)  // Set up the GET variables  
 [484:](#2kicxvv)  $this->\_get\_args = array\_merge($this->\_get\_args, $this->uri->ruri\_to\_assoc());  
 [485:](#znn83o)   
 [486:](#3jnaqrh)  // Try to find a format for the request (means we have a request body)  
 [487:](#1ysl0za)  $this->request->format = $this->\_detect\_input\_format();  
 [488:](#4is8jn3)   
 [489:](#2xxituw)  // Not all methods have a body attached with them  
 [490:](#1d2t42p)  $this->request->body = NULL;  
 [491:](#3x2gmqi)   
 [492:](#2c7qwyb)  $this->{'\_parse\_' . $this->request->method}();  
 [493:](#rd1764)   
 [494:](#3bcoptx)  // Now we know all about our request, let's try and parse the body if it exists  
 [495:](#1qhz01q)  if ($this->request->format && $this->request->body)  
 [496:](#4ahmipj)  {  
 [497:](#2pmwsxc)  $this->request->body = $this->format->factory($this->request->body, $this->request->format)->to\_array();  
 [498:](#14s7355)  // Assign payload arguments to proper method container  
 [499:](#3orulsy)  $this->{'\_'.$this->request->method.'\_args'} = $this->request->body;  
 [500:](#23x4w0r)  }  
 [501:](#j2f68k)   
 [502:](#3322owd)  // Merge both for one mega-args variable  
 [503:](#1i7cz46)  $this->\_args = array\_merge(  
 [504:](#4270hrz)  $this->\_get\_args,  
 [505:](#2hcarzs)  $this->\_options\_args,  
 [506:](#whl27l)  $this->\_patch\_args,  
 [507:](#3gh8kve)  $this->\_head\_args,  
 [508:](#1vmiv37)  $this->\_put\_args,  
 [509:](#4fm6dr0)  $this->\_post\_args,  
 [510:](#2urgnyt)  $this->\_delete\_args,  
 [511:](#19wqy6m)  $this->{'\_'.$this->request->method.'\_args'}  
 [512:](#3tweguf)  );  
 [513:](#291or28)   
 [514:](#o6z1a1)  // Which format should the data be returned in?  
 [515:](#386mjxu)  $this->response->format = $this->\_detect\_output\_format();  
 [516:](#1nbwu5n)   
 [517:](#47bkctg)  // Which language should the data be returned in?  
 [518:](#2mgun19)  $this->response->lang = $this->\_detect\_lang();  
 [519:](#11m4x92)   
 [520:](#3llsfwv)  // Extend this function to apply additional checking early on in the process  
 [521:](#20r2q4o)  $this->early\_checks();  
 [522:](#4kqq8sh)   
 [523:](#2zw0j0a)  // Load DB if its enabled  
 [524:](#1f1at83)  if ($this->config->item('rest\_database\_group') && ($this->config->item('rest\_enable\_keys') || $this->config->item('rest\_enable\_logging')))  
 [525:](#3z0ybvw)  {  
 [526:](#2e68m3p)  $this->rest->db = $this->load->database($this->config->item('rest\_database\_group'), TRUE);  
 [527:](#tbiwbi)  }  
 [528:](#3db6ezb)   
 [529:](#1sggp74)  // Use whatever database is in use (isset returns FALSE)  
 [530:](#4cg47ux)  elseif (property\_exists($this, 'db'))  
 [531:](#2rlei2q)  {  
 [532:](#16qosaj)  $this->rest->db = $this->db;  
 [533:](#3qqcayc)  }  
 [534:](#25vml65)   
 [535:](#l0wvdy)  // Check if there is a specific auth type for the current class/method  
 [536:](#350ke1r)  // \_auth\_override\_check could exit so we need $this->rest->db initialized before  
 [537:](#1k5uo9k)  $this->auth\_override = $this->\_auth\_override\_check();  
 [538:](#445i6xd)   
 [539:](#2jash56)  // Checking for keys? GET TO WorK!  
 [540:](#yg2rcz)  // Skip keys test for $config['auth\_override\_class\_method']['class'['method'] = 'none'  
 [541:](#3ifqa0s)  if ($this->config->item('rest\_enable\_keys') && $this->auth\_override !== TRUE)  
 [542:](#1xl0k8l)  {  
 [543:](#4hko2we)  $this->\_allow = $this->\_detect\_api\_key();  
 [544:](#2wpyd47)  }  
 [545:](#1bv8nc0)   
 [546:](#3vuw5zt)  // Only allow ajax requests  
 [547:](#2b06g7m)  if ($this->input->is\_ajax\_request() === FALSE && $this->config->item('rest\_ajax\_only'))  
 [548:](#q5gqff)  {  
 [549:](#3a54938)  // Display an error response  
 [550:](#1paejb1)  $this->response([  
 [551:](#49a21yu)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [552:](#2ofcc6n)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_ajax\_only')  
 [553:](#13kmmeg)  ], self::HTTP\_NOT\_ACCEPTABLE);  
 [554:](#3nka529)  }  
 [555:](#22pkfa2)   
 [556:](#huuphv)  // When there is no specific override for the current class/method, use the default auth value set in the config  
 [557:](#31ui85o)  if ($this->auth\_override === FALSE && ! ($this->config->item('rest\_enable\_keys') && $this->\_allow === TRUE) || ($this->config->item('allow\_auth\_and\_keys') === TRUE && $this->\_allow === TRUE))  
 [558:](#1gzsidh)  {  
 [559:](#40zg11a)  $rest\_auth = strtolower($this->config->item('rest\_auth'));  
 [560:](#2g4qb93)  switch ($rest\_auth)  
 [561:](#va0lgw)  {  
 [562:](#3f9o44p)  case 'basic':  
 [563:](#1ueyeci)  $this->\_prepare\_basic\_auth();  
 [564:](#4eelx0b)  break;  
 [565:](#2tjw784)  case 'digest':  
 [566:](#18p6hfx)  $this->\_prepare\_digest\_auth();  
 [567:](#3sou03q)  break;  
 [568:](#27u4abj)  case 'session':  
 [569:](#mzekjc)  $this->\_check\_php\_session();  
 [570:](#36z2375)  break;  
 [571:](#1m4cdey)  }  
 [572:](#463zw2r)  if ($this->config->item('rest\_ip\_whitelist\_enabled') === TRUE)  
 [573:](#2l9a6ak)  {  
 [574:](#10ekgid)  $this->\_check\_whitelist\_auth();  
 [575:](#3ke7z66)  }  
 [576:](#1zji9dz)  }  
 [577:](#4jj5s1s)  }  
 [578:](#2yog29l)   
 [579:](#1dtqche)  /\*\*  
 [580:](#3xtdv57)  \* Deconstructor  
 [581:](#2cyo5d0)  \*  
 [582:](#s3yfkt)  \* @author Chris Kacerguis  
 [583:](#3c3ly8m)  \* @access public  
 [584:](#1r8w8gf)  \* @return void  
 [585:](#4b8jr48)  \*/  
 [586:](#2qdu1c1)  public function \_\_destruct()  
 [587:](#15j4bju)  {  
 [588:](#3piru7n)  // Get the current timestamp  
 [589:](#24o24fg)  $this->\_end\_rtime = microtime(TRUE);  
 [590:](#jtcen9)   
 [591:](#33szxb2)  // Log the loading time to the log table  
 [592:](#1iya7iv)  if ($this->config->item('rest\_enable\_logging') === TRUE)  
 [593:](#42xxq6o)  {  
 [594:](#2i380eh)  $this->\_log\_access\_time();  
 [595:](#x8iama)  }  
 [596:](#3h85ta3)  }  
 [597:](#1wdg3hw)   
 [598:](#4gd3m5p)  /\*\*  
 [599:](#2vidwdi)  \* Requests are not made to methods directly, the request will be for  
 [600:](#1ano6lb)  \* an "object". This simply maps the object and method to the correct  
 [601:](#3unbp94)  \* Controller method  
 [602:](#29slzgx)  \*  
 [603:](#oxw9oq)  \* @access public  
 [604:](#38xjscj)  \* @param string $object\_called  
 [605:](#1o2u2kc)  \* @param array $arguments The arguments passed to the controller method  
 [606:](#482hl85)  \*/  
 [607:](#2n7rvfy)  public function \_remap($object\_called, $arguments = [])  
 [608:](#12d25nr)  {  
 [609:](#3mcpobk)  // Should we answer if not over SSL?  
 [610:](#21hzyjd)  if ($this->config->item('force\_https') && $this->request->ssl === FALSE)  
 [611:](#gna8r6)  {  
 [612:](#30mxrez)  $this->response([  
 [613:](#1fs81ms)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [614:](#3zrvkal)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_unsupported')  
 [615:](#2ex5uie)  ], self::HTTP\_FORBIDDEN);  
 [616:](#u2g4q7)  }  
 [617:](#3e23ne0)   
 [618:](#1t7dxlt)  // Remove the supported format from the function name e.g. index.json => index  
 [619:](#4d71g9m)  $object\_called = preg\_replace('/^(.\*)\.(?:'.implode('|', array\_keys($this->\_supported\_formats)).')$/', '$1', $object\_called);  
 [620:](#2scbqhf)   
 [621:](#17hm0p8)  $controller\_method = $object\_called.'\_'.$this->request->method;  
 [622:](#3rh9jd1)   
 [623:](#26mjtku)  // Do we want to log this method (if allowed by config)?  
 [624:](#lru3sn)  $log\_method = ! (isset($this->methods[$controller\_method]['log']) && $this->methods[$controller\_method]['log'] === FALSE);  
 [625:](#35rhmgg)   
 [626:](#1kwrwo9)  // Use keys for this method?  
 [627:](#44wffc2)  $use\_key = ! (isset($this->methods[$controller\_method]['key']) && $this->methods[$controller\_method]['key'] === FALSE);  
 [628:](#2k1ppjv)   
 [629:](#z6zzro)  // They provided a key, but it wasn't valid, so get them out of here  
 [630:](#3j6nifh)  if ($this->config->item('rest\_enable\_keys') && $use\_key && $this->\_allow === FALSE)  
 [631:](#1ybxsna)  {  
 [632:](#4iblbb3)  if ($this->config->item('rest\_enable\_logging') && $log\_method)  
 [633:](#2xgvliw)  {  
 [634:](#1cm5vqp)  $this->\_log\_request();  
 [635:](#3wlteei)  }  
 [636:](#2br3omb)   
 [637:](#qwdyu4)  $this->response([  
 [638:](#3aw1hhx)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [639:](#1q1brpq)  $this->config->item('rest\_message\_field\_name') => sprintf($this->lang->line('text\_rest\_invalid\_api\_key'), $this->rest->key)  
 [640:](#4a0zadj)  ], self::HTTP\_FORBIDDEN);  
 [641:](#2p69klc)  }  
 [642:](#14bjut5)   
 [643:](#3ob7dgy)  // Check to see if this key has access to the requested controller  
 [644:](#23ghnor)  if ($this->config->item('rest\_enable\_keys') && $use\_key && empty($this->rest->key) === FALSE && $this->\_check\_access() === FALSE)  
 [645:](#ilrxwk)  {  
 [646:](#32lfgkd)  if ($this->config->item('rest\_enable\_logging') && $log\_method)  
 [647:](#1hqpqs6)  {  
 [648:](#41qd9fz)  $this->\_log\_request();  
 [649:](#2gvnjns)  }  
 [650:](#w0xtvl)   
 [651:](#3g0lcje)  $this->response([  
 [652:](#1v5vmr7)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [653:](#4f5j5f0)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_api\_key\_unauthorized')  
 [654:](#2uatfmt)  ], self::HTTP\_UNAUTHORIZED);  
 [655:](#19g3pum)  }  
 [656:](#3tfr8if)   
 [657:](#28l1iq8)  // Sure it exists, but can they do anything with it?  
 [658:](#nqbsy1)  if (method\_exists($this, $controller\_method) === FALSE)  
 [659:](#37pzblu)  {  
 [660:](#1mv9ltn)  $this->response([  
 [661:](#46ux4hg)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [662:](#2m07ep9)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_unknown\_method')  
 [663:](#115hox2)  ], self::HTTP\_NOT\_FOUND);  
 [664:](#3l557kv)  }  
 [665:](#20afhso)   
 [666:](#4ka30gh)  // Doing key related stuff? Can only do it if they have a key right?  
 [667:](#2zfdaoa)  if ($this->config->item('rest\_enable\_keys') && empty($this->rest->key) === FALSE)  
 [668:](#1eknkw3)  {  
 [669:](#3ykb3jw)  // Check the limit  
 [670:](#2dpldrp)  if ($this->config->item('rest\_enable\_limits') && $this->\_check\_limit($controller\_method) === FALSE)  
 [671:](#suvnzi)  {  
 [672:](#3cuj6nb)  $response = [$this->config->item('rest\_status\_field\_name') => FALSE, $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_api\_key\_time\_limit')];  
 [673:](#1rztgv4)  $this->response($response, self::HTTP\_UNAUTHORIZED);  
 [674:](#4bzgzix)  }  
 [675:](#2r4r9qq)   
 [676:](#16a1jyj)  // If no level is set use 0, they probably aren't using permissions  
 [677:](#3q9p2mc)  $level = isset($this->methods[$controller\_method]['level']) ? $this->methods[$controller\_method]['level'] : 0;  
 [678:](#25ezcu5)   
 [679:](#kk9n1y)  // If no level is set, or it is lower than/equal to the key's level  
 [680:](#34jx5pr)  $authorized = $level <= $this->rest->level;  
 [681:](#1jp7fxk)  // IM TELLIN!  
 [682:](#43ouyld)  if ($this->config->item('rest\_enable\_logging') && $log\_method)  
 [683:](#2iu58t6)  {  
 [684:](#xzfj0z)  $this->\_log\_request($authorized);  
 [685:](#3hz31os)  }  
 [686:](#1x4dbwl)  if($authorized === FALSE)  
 [687:](#4h40uke)  {  
 [688:](#2w9b4s7)  // They don't have good enough perms  
 [689:](#1belf00)  $response = [$this->config->item('rest\_status\_field\_name') => FALSE, $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_api\_key\_permissions')];  
 [690:](#3ve8xnt)  $this->response($response, self::HTTP\_UNAUTHORIZED);  
 [691:](#2ajj7vm)  }  
 [692:](#poti3f)  }  
 [693:](#39oh0r8)   
 [694:](#1otraz1)  // No key stuff, but record that stuff is happening  
 [695:](#48tetmu)  elseif ($this->config->item('rest\_enable\_logging') && $log\_method)  
 [696:](#2nyp3un)  {  
 [697:](#133ze2g)  $this->\_log\_request($authorized = TRUE);  
 [698:](#3n3mwq9)  }  
 [699:](#228x6y2)   
 [700:](#he7h5v)  // Call the controller method and passed arguments  
 [701:](#31duzto)  try  
 [702:](#1gj5a1h)  {  
 [703:](#40isspa)  call\_user\_func\_array([$this, $controller\_method], $arguments);  
 [704:](#2fo32x3)  }  
 [705:](#utdd4w)  catch (Exception $ex)  
 [706:](#3et0vsp)  {  
 [707:](#1tyb60i)  // If the method doesn't exist, then the error will be caught and an error response shown  
 [708:](#4dxyoob)  $this->response([  
 [709:](#2t38yw4)  $this->config->item('rest\_status\_field\_name') => FALSE,  
 [710:](#188j93x)  $this->config->item('rest\_message\_field\_name') => [  
 [711:](#3s86rrq)  'classname' => get\_class($ex),  
 [712:](#27dh1zj)  'message' => $ex->getMessage()  
 [713:](#mirc7c)  ]  
 [714:](#36ieuv5)  ], self::HTTP\_INTERNAL\_SERVER\_ERROR);  
 [715:](#1lnp52y)  }  
 [716:](#45ncnqr)  }  
 [717:](#2ksmxyk)   
 [718:](#zxx86d)  /\*\*  
 [719:](#3jxkqu6)  \* Takes mixed data and optionally a status code, then creates the response  
 [720:](#1z2v11z)  \*  
 [721:](#4j2ijps)  \* @access public  
 [722:](#2y7stxl)  \* @param array|NULL $data Data to output to the user  
 [723:](#1dd345e)  \* @param int|NULL $http\_code HTTP status code  
 [724:](#3xcqmt7)  \* @param bool $continue TRUE to flush the response to the client and continue  
 [725:](#2ci0x10)  \* running the script; otherwise, exit  
 [726:](#rnb78t)  \*/  
 [727:](#3bmypwm)  public function response($data = NULL, $http\_code = NULL, $continue = FALSE)  
 [728:](#1qs904f)  {  
 [729:](#4arwis8)  // If the HTTP status is not NULL, then cast as an integer  
 [730:](#2px6t01)  if ($http\_code !== NULL)  
 [731:](#152h37u)  {  
 [732:](#3p24lvn)  // So as to be safe later on in the process  
 [733:](#247ew3g)  $http\_code = (int) $http\_code;  
 [734:](#jcp6b9)  }  
 [735:](#33ccoz2)   
 [736:](#1ihmz6v)  // Set the output as NULL by default  
 [737:](#42hahuo)  $output = NULL;  
 [738:](#2hmks2h)   
 [739:](#wrv2aa)  // If data is NULL and no HTTP status code provided, then display, error and exit  
 [740:](#3griky3)  if ($data === NULL && $http\_code === NULL)  
 [741:](#1vwsv5w)  {  
 [742:](#4fwgdtp)  $http\_code = self::HTTP\_NOT\_FOUND;  
 [743:](#2v1qo1i)  }  
 [744:](#1a70y9b)   
 [745:](#3u6ogx4)  // If data is not NULL and a HTTP status code provided, then continue  
 [746:](#29byr4x)  elseif ($data !== NULL)  
 [747:](#oh91cq)  {  
 [748:](#38gwk0j)  // If the format method exists, call and return the output in that format  
 [749:](#1nm6u8c)  if (method\_exists($this->format, 'to\_' . $this->response->format))  
 [750:](#47lucw5)  {  
 [751:](#2mr4n3y)  // Set the format header  
 [752:](#11wexbr)  $this->output->set\_content\_type($this->\_supported\_formats[$this->response->format], strtolower($this->config->item('charset')));  
 [753:](#3lw2fzk)  $output = $this->format->factory($data)->{'to\_' . $this->response->format}();  
 [754:](#211cq7d)   
 [755:](#4l108v6)  // An array must be parsed as a string, so as not to cause an array to string error  
 [756:](#306aj2z)  // Json is the most appropriate form for such a datatype  
 [757:](#1fbktas)  if ($this->response->format === 'array')  
 [758:](#3zb8byl)  {  
 [759:](#2egim6e)  $output = $this->format->factory($output)->{'to\_json'}();  
 [760:](#tlswe7)  }  
 [761:](#3dlgf20)  }  
 [762:](#1sqqp9t)  else  
 [763:](#4cqe7xm)  {  
 [764:](#2rvoi5f)  // If an array or object, then parse as a json, so as to be a 'string'  
 [765:](#170ysd8)  if (is\_array($data) || is\_object($data))  
 [766:](#3r0mb11)  {  
 [767:](#265wl8u)  $data = $this->format->factory($data)->{'to\_json'}();  
 [768:](#lb6vgn)  }  
 [769:](#35aue4g)   
 [770:](#1kg4oc9)  // Format is not supported, so output the raw data as a string  
 [771:](#44fs702)  $output = $data;  
 [772:](#2jl2h7v)  }  
 [773:](#yqcrfo)  }  
 [774:](#3iq0a3h)   
 [775:](#1xvakba)  // If not greater than zero, then set the HTTP status code as 200 by default  
 [776:](#4huy2z3)  // Though perhaps 500 should be set instead, for the developer not passing a  
 [777:](#2x08d6w)  // correct HTTP status code  
 [778:](#1c5inep)  $http\_code > 0 || $http\_code = self::HTTP\_OK;  
 [779:](#3w5662i)   
 [780:](#2baggab)  $this->output->set\_status\_header($http\_code);  
 [781:](#qfqqi4)   
 [782:](#3afe95x)  // JC: Log response code only if rest logging enabled  
 [783:](#1pkojdq)  if ($this->config->item('rest\_enable\_logging') === TRUE)  
 [784:](#49kc21j)  {  
 [785:](#2opmc9c)  $this->\_log\_response\_code($http\_code);  
 [786:](#13uwmh5)  }  
 [787:](#3nuk54y)   
 [788:](#22zufcr)  // Output the data  
 [789:](#i54pkk)  $this->output->set\_output($output);  
 [790:](#324s88d)   
 [791:](#1ha2ig6)  if ($continue === FALSE)  
 [792:](#419q13z)  {  
 [793:](#2gf0bbs)  // Display the data and exit execution  
 [794:](#vkaljl)  $this->output->\_display();  
 [795:](#3fjy47e)  exit;  
 [796:](#1up8ef7)  }  
 [797:](#4eovx30)   
 [798:](#2tu67at)  // Otherwise dump the output automatically  
 [799:](#18zghim)  }  
 [800:](#3sz406f)   
 [801:](#284eae8)  /\*\*  
 [802:](#n9okm1)  \* Takes mixed data and optionally a status code, then creates the response  
 [803:](#379c39u)  \* within the buffers of the Output class. The response is sent to the client  
 [804:](#1memdhn)  \* lately by the framework, after the current controller's method termination.  
 [805:](#46e9w5g)  \* All the hooks after the controller's method termination are executable  
 [806:](#2ljk6d9)  \*  
 [807:](#10ougl2)  \* @access public  
 [808:](#3kohz8v)  \* @param array|NULL $data Data to output to the user  
 [809:](#1zts9go)  \* @param int|NULL $http\_code HTTP status code  
 [810:](#4jtfs4h)  \*/  
 [811:](#2yyq2ca)  public function set\_response($data = NULL, $http\_code = NULL)  
 [812:](#1e40ck3)  {  
 [813:](#3y3nv7w)  $this->response($data, $http\_code, TRUE);  
 [814:](#2d8y5fp)  }  
 [815:](#se8fni)   
 [816:](#3cdvybb)  /\*\*  
 [817:](#1rj68j4)  \* Get the input format e.g. json or xml  
 [818:](#4bitr6x)  \*  
 [819:](#2qo41eq)  \* @access protected  
 [820:](#15tebmj)  \* @return string|NULL Supported input format; otherwise, NULL  
 [821:](#3pt1uac)  \*/  
 [822:](#24yc4i5)  protected function \_detect\_input\_format()  
 [823:](#k3mepy)  {  
 [824:](#3439xdr)  // Get the CONTENT-TYPE value from the SERVER variable  
 [825:](#1j8k7lk)  $content\_type = $this->input->server('CONTENT\_TYPE');  
 [826:](#4387q9d)   
 [827:](#2idi0h6)  if (empty($content\_type) === FALSE)  
 [828:](#xisaoz)  {  
 [829:](#3hiftcs)  // If a semi-colon exists in the string, then explode by ; and get the value of where  
 [830:](#1wnq3kl)  // the current array pointer resides. This will generally be the first element of the array  
 [831:](#4gndm8e)  $content\_type = (strpos($content\_type, ';') !== FALSE ? current(explode(';', $content\_type)) : $content\_type);  
 [832:](#2vsnwg7)   
 [833:](#1axy6o0)  // Check all formats against the CONTENT-TYPE header  
 [834:](#3uxlpbt)  foreach ($this->\_supported\_formats as $type => $mime)  
 [835:](#2a2vzjm)  {  
 [836:](#p869rf)  // $type = format e.g. csv  
 [837:](#397tsf8)  // $mime = mime type e.g. application/csv  
 [838:](#1od42n1)   
 [839:](#48crlau)  // If both the mime types match, then return the format  
 [840:](#2ni1vin)  if ($content\_type === $mime)  
 [841:](#12nc5qg)  {  
 [842:](#3mmzoe9)  return $type;  
 [843:](#21s9ym2)  }  
 [844:](#gxk8tv)  }  
 [845:](#30x7rho)  }  
 [846:](#1g2i1ph)   
 [847:](#4025kda)  return NULL;  
 [848:](#2f7ful3)  }  
 [849:](#ucq4sw)   
 [850:](#3ecdngp)  /\*\*  
 [851:](#1thnxoi)  \* Gets the default format from the configuration. Fallbacks to 'json'  
 [852:](#4dhbgcb)  \* if the corresponding configuration option $config['rest\_default\_format']  
 [853:](#2smlqk4)  \* is missing or is empty  
 [854:](#17rw0rx)  \*  
 [855:](#3rrjjfq)  \* @access protected  
 [856:](#26wttnj)  \* @return string The default supported input format  
 [857:](#m243vc)  \*/  
 [858:](#361rmj5)  protected function \_get\_default\_output\_format()  
 [859:](#1l71wqy)  {  
 [860:](#456pfer)  $default\_format = (string) $this->config->item('rest\_default\_format');  
 [861:](#2kbzpmk)  return $default\_format === '' ? 'json' : $default\_format;  
 [862:](#zh9zud)  }  
 [863:](#3jgxii6)   
 [864:](#1ym7spz)  /\*\*  
 [865:](#4ilvbds)  \* Detect which format should be used to output the data  
 [866:](#2xr5lll)  \*  
 [867:](#1cwfvte)  \* @access protected  
 [868:](#3ww3eh7)  \* @return mixed|NULL|string Output format  
 [869:](#2c1dop0)  \*/  
 [870:](#r6nywt)  protected function \_detect\_output\_format()  
 [871:](#3b6bhkm)  {  
 [872:](#1qblrsf)  // Concatenate formats to a regex pattern e.g. \.(csv|json|xml)  
 [873:](#4ab9ag8)  $pattern = '/\.('.implode('|', array\_keys($this->\_supported\_formats)).')($|\/)/';  
 [874:](#2pgjko1)  $matches = [];  
 [875:](#14ltuvu)   
 [876:](#3olhdjn)  // Check if a file extension is used e.g. http://example.com/api/index.json?param1=param2  
 [877:](#23qrnrg)  if (preg\_match($pattern, $this->uri->uri\_string(), $matches))  
 [878:](#iw1xz9)  {  
 [879:](#32vpgn2)  return $matches[1];  
 [880:](#1i0zquv)  }  
 [881:](#420n9io)   
 [882:](#2h5xjqh)  // Get the format parameter named as 'format'  
 [883:](#wb7tya)  if (isset($this->\_get\_args['format']))  
 [884:](#3gavcm3)  {  
 [885:](#1vg5mtw)  $format = strtolower($this->\_get\_args['format']);  
 [886:](#4fft5hp)   
 [887:](#2ul3fpi)  if (isset($this->\_supported\_formats[$format]) === TRUE)  
 [888:](#19qdpxb)  {  
 [889:](#3tq18l4)  return $format;  
 [890:](#28vbisx)  }  
 [891:](#o0lt0q)  }  
 [892:](#3809boj)   
 [893:](#1n5jlwc)  // Get the HTTP\_ACCEPT server variable  
 [894:](#47574k5)  $http\_accept = $this->input->server('HTTP\_ACCEPT');  
 [895:](#2mahery)   
 [896:](#11frozr)  // Otherwise, check the HTTP\_ACCEPT server variable  
 [897:](#3lff7nk)  if ($this->config->item('rest\_ignore\_http\_accept') === FALSE && $http\_accept !== NULL)  
 [898:](#20kphvd)  {  
 [899:](#4kkd0j6)  // Check all formats against the HTTP\_ACCEPT header  
 [900:](#2zpnaqz)  foreach (array\_keys($this->\_supported\_formats) as $format)  
 [901:](#1euxkys)  {  
 [902:](#3yul3ml)  // Has this format been requested?  
 [903:](#2dzvdue)  if (strpos($http\_accept, $format) !== FALSE)  
 [904:](#t55o27)  {  
 [905:](#3d4t6q0)  if ($format !== 'html' && $format !== 'xml')  
 [906:](#1sa3gxt)  {  
 [907:](#4c9qzlm)  // If not HTML or XML assume it's correct  
 [908:](#2rf19tf)  return $format;  
 [909:](#16kbk18)  }  
 [910:](#3qjz2p1)  elseif ($format === 'html' && strpos($http\_accept, 'xml') === FALSE)  
 [911:](#25p9cwu)  {  
 [912:](#kujn4n)  // HTML or XML have shown up as a match  
 [913:](#34u75sg)  // If it is truly HTML, it wont want any XML  
 [914:](#1jzhg09)  return $format;  
 [915:](#43z4yo2)  }  
 [916:](#2j4f8vv)  else if ($format === 'xml' && strpos($http\_accept, 'html') === FALSE)  
 [917:](#y9pj3o)  {  
 [918:](#3i9d1rh)  // If it is truly XML, it wont want any HTML  
 [919:](#1xenbza)  return $format;  
 [920:](#4heaun3)  }  
 [921:](#2wjl4uw)  }  
 [922:](#1bovf2p)  }  
 [923:](#3voixqi)  }  
 [924:](#2att7yb)   
 [925:](#pz3i64)  // Check if the controller has a default format  
 [926:](#39yr0tx)  if (empty($this->rest\_format) === FALSE)  
 [927:](#1p41b1q)  {  
 [928:](#493otpj)  return $this->rest\_format;  
 [929:](#2o8z3xc)  }  
 [930:](#13e9e55)   
 [931:](#3ndwwsy)  // Obtain the default format from the configuration  
 [932:](#22j770r)  return $this->\_get\_default\_output\_format();  
 [933:](#hohh8k)  }  
 [934:](#31o4zwd)   
 [935:](#1gtfa46)  /\*\*  
 [936:](#40t2srz)  \* Get the HTTP request string e.g. get or post  
 [937:](#2fyd2zs)  \*  
 [938:](#v3nd7l)  \* @access protected  
 [939:](#3f3avve)  \* @return string|NULL Supported request method as a lowercase string; otherwise, NULL if not supported  
 [940:](#1u8l637)  \*/  
 [941:](#4e88or0)  protected function \_detect\_method()  
 [942:](#2tdiyyt)  {  
 [943:](#18it96m)  // Declare a variable to store the method  
 [944:](#3sigruf)  $method = NULL;  
 [945:](#27nr228)   
 [946:](#mt1ca1)  // Determine whether the 'enable\_emulate\_request' setting is enabled  
 [947:](#36souxu)  if ($this->config->item('enable\_emulate\_request') === TRUE)  
 [948:](#1lxz55n)  {  
 [949:](#45xmntg)  $method = $this->input->post('\_method');  
 [950:](#2l2wy19)  if ($method === NULL)  
 [951:](#1087892)  {  
 [952:](#3k7uqwv)  $method = $this->input->server('HTTP\_X\_HTTP\_METHOD\_OVERRIDE');  
 [953:](#1zd514o)  }  
 [954:](#4jcsjsh)   
 [955:](#2yi2u0a)  $method = strtolower($method);  
 [956:](#1dnd483)  }  
 [957:](#3xn0mvw)   
 [958:](#2csax3p)  if (empty($method))  
 [959:](#rxl7bi)  {  
 [960:](#3bx8pzb)  // Get the request method as a lowercase string  
 [961:](#1r2j074)  $method = $this->input->method();  
 [962:](#4b26iux)  }  
 [963:](#2q7gt2q)   
 [964:](#15cr3aj)  return in\_array($method, $this->allowed\_http\_methods) && method\_exists($this, '\_parse\_' . $method) ? $method : 'get';  
 [965:](#3pcelyc)  }  
 [966:](#24how65)   
 [967:](#jmz6dy)  /\*\*  
 [968:](#33mmp1r)  \* See if the user has provided an API key  
 [969:](#1irwz9k)  \*  
 [970:](#42rkhxd)  \* @access protected  
 [971:](#2hwus56)  \* @return bool  
 [972:](#x252cz)  \*/  
 [973:](#3h1sl0s)  protected function \_detect\_api\_key()  
 [974:](#1w72v8l)  {  
 [975:](#4g6qdwe)  // Get the api key name variable set in the rest config file  
 [976:](#2vc0o47)  $api\_key\_variable = $this->config->item('rest\_key\_name');  
 [977:](#1ahayc0)   
 [978:](#3ugygzt)  // Work out the name of the SERVER entry based on config  
 [979:](#29m8r7m)  $key\_name = 'HTTP\_' . strtoupper(str\_replace('-', '\_', $api\_key\_variable));  
 [980:](#orj1ff)   
 [981:](#38r6k38)  $this->rest->key = NULL;  
 [982:](#1nwgub1)  $this->rest->level = NULL;  
 [983:](#47w4cyu)  $this->rest->user\_id = NULL;  
 [984:](#2n1en6n)  $this->rest->ignore\_limits = FALSE;  
 [985:](#126oxeg)   
 [986:](#3m6cg29)  // Find the key from server or arguments  
 [987:](#21bmqa2)  if (($key = isset($this->\_args[$api\_key\_variable]) ? $this->\_args[$api\_key\_variable] : $this->input->server($key\_name)))  
 [988:](#4lba8xv)  {  
 [989:](#30gkj5o)  if ( ! ($row = $this->rest->db->where($this->config->item('rest\_key\_column'), $key)->get($this->config->item('rest\_keys\_table'))->row()))  
 [990:](#1flutdh)  {  
 [991:](#3zlic1a)  return FALSE;  
 [992:](#2eqsm93)  }  
 [993:](#tw2wgw)   
 [994:](#3dvqf4p)  $this->rest->key = $row->{$this->config->item('rest\_key\_column')};  
 [995:](#1t10pci)   
 [996:](#4d0o80b)  isset($row->user\_id) && $this->rest->user\_id = $row->user\_id;  
 [997:](#2s5yi84)  isset($row->level) && $this->rest->level = $row->level;  
 [998:](#17b8sfx)  isset($row->ignore\_limits) && $this->rest->ignore\_limits = $row->ignore\_limits;  
 [999:](#3rawb3q)   
[1000:](#26g6lbj)  $this->\_apiuser = $row;  
[1001:](#llgvjc)   
[1002:](#35l4e75)  /\*  
[1003:](#1kqeoey)  \* If "is private key" is enabled, compare the ip address with the list  
[1004:](#44q272r)  \* of valid ip addresses stored in the database  
[1005:](#2jvchak)  \*/  
[1006:](#z0mrid)  if (empty($row->is\_private\_key) === FALSE)  
[1007:](#3j0aa66)  {  
[1008:](#1y5kkdz)  // Check for a list of valid ip addresses  
[1009:](#4i5831s)  if (isset($row->ip\_addresses))  
[1010:](#2xaid9l)  {  
[1011:](#1cfsnhe)  // multiple ip addresses must be separated using a comma, explode and loop  
[1012:](#3wfg657)  $list\_ip\_addresses = explode(',', $row->ip\_addresses);  
[1013:](#2bkqgd0)  $found\_address = FALSE;  
[1014:](#qq0qkt)   
[1015:](#3apo98m)  foreach ($list\_ip\_addresses as $ip\_address)  
[1016:](#1puyjgf)  {  
[1017:](#49um248)  if ($this->input->ip\_address() === trim($ip\_address))  
[1018:](#2ozwcc1)  {  
[1019:](#1456mju)  // there is a match, set the the value to TRUE and break out of the loop  
[1020:](#3o4u57n)  $found\_address = TRUE;  
[1021:](#23a4ffg)  break;  
[1022:](#ifepn9)  }  
[1023:](#32f28b2)  }  
[1024:](#1hkciiv)   
[1025:](#41k016o)  return $found\_address;  
[1026:](#2gpabeh)  }  
[1027:](#vuklma)  else  
[1028:](#3fu84a3)  {  
[1029:](#1uziehw)  // There should be at least one IP address for this private key  
[1030:](#4ez5x5p)  return FALSE;  
[1031:](#2u4g7di)  }  
[1032:](#199qhlb)  }  
[1033:](#3t9e094)   
[1034:](#28eoagx)  return TRUE;  
[1035:](#njykoq)  }  
[1036:](#37jm3cj)   
[1037:](#1mowdkc)  // No key has been sent  
[1038:](#46ojw85)  return FALSE;  
[1039:](#2ltu6fy)  }  
[1040:](#10z4gnr)   
[1041:](#3kyrzbk)  /\*\*  
[1042:](#20429jd)  \* Preferred return language  
[1043:](#4k3ps76)  \*  
[1044:](#2z902ez)  \* @access protected  
[1045:](#1eeacms)  \* @return string|NULL The language code  
[1046:](#3ydxval)  \*/  
[1047:](#2dj85ie)  protected function \_detect\_lang()  
[1048:](#soifq7)  {  
[1049:](#3co5ye0)  $lang = $this->input->server('HTTP\_ACCEPT\_LANGUAGE');  
[1050:](#1rtg8lt)  if ($lang === NULL)  
[1051:](#4bt3r9m)  {  
[1052:](#2qye1hf)  return NULL;  
[1053:](#163obp8)  }  
[1054:](#3q3bud1)   
[1055:](#258m4ku)  // It appears more than one language has been sent using a comma delimiter  
[1056:](#kdwesn)  if (strpos($lang, ',') !== FALSE)  
[1057:](#34djxgg)  {  
[1058:](#1jiu7o9)  $langs = explode(',', $lang);  
[1059:](#43ihqc2)   
[1060:](#2ins0jv)  $return\_langs = [];  
[1061:](#xt2aro)  foreach ($langs as $lang)  
[1062:](#3hsptfh)  {  
[1063:](#1wy03na)  // Remove weight and trim leading and trailing whitespace  
[1064:](#4gxnmb3)  list($lang) = explode(';', $lang);  
[1065:](#2w2xwiw)  $return\_langs[] = trim($lang);  
[1066:](#1b886qp)  }  
[1067:](#3v7vpei)   
[1068:](#2ad5zmb)  return $return\_langs;  
[1069:](#pig9u4)  }  
[1070:](#39i3shx)   
[1071:](#1one2pq)  // Otherwise simply return as a string  
[1072:](#48n1ldj)  return $lang;  
[1073:](#2nsbvlc)  }  
[1074:](#12xm5t5)   
[1075:](#3mx9ogy)  /\*\*  
[1076:](#222jyor)  \* Add the request to the log table  
[1077:](#h7u8wk)  \*  
[1078:](#317hrkd)  \* @access protected  
[1079:](#1gcs1s6)  \* @param bool $authorized TRUE the user is authorized; otherwise, FALSE  
[1080:](#40cfkfz)  \* @return bool TRUE the data was inserted; otherwise, FALSE  
[1081:](#2fhpuns)  \*/  
[1082:](#un04vl)  protected function \_log\_request($authorized = FALSE)  
[1083:](#3emnnje)  {  
[1084:](#1trxxr7)  // Insert the request into the log table  
[1085:](#4drlgf0)  $is\_inserted = $this->rest->db  
[1086:](#2swvqmt)  ->insert(  
[1087:](#18260um)  $this->config->item('rest\_logs\_table'), [  
[1088:](#3s1tjif)  'uri' => $this->uri->uri\_string(),  
[1089:](#2773tq8)  'method' => $this->request->method,  
[1090:](#mce3y1)  'params' => $this->\_args ? ($this->config->item('rest\_logs\_json\_params') === TRUE ? json\_encode($this->\_args) : serialize($this->\_args)) : NULL,  
[1091:](#36c1mlu)  'api\_key' => isset($this->rest->key) ? $this->rest->key : '',  
[1092:](#1lhbwtn)  'ip\_address' => $this->input->ip\_address(),  
[1093:](#45gzfhg)  'time' => time(),  
[1094:](#2km9pp9)  'authorized' => $authorized  
[1095:](#zrjzx2)  ]);  
[1096:](#3jr7ikv)   
[1097:](#1ywhsso)  // Get the last insert id to update at a later stage of the request  
[1098:](#4iw5bgh)  $this->\_insert\_id = $this->rest->db->insert\_id();  
[1099:](#2y1floa)   
[1100:](#1d6pvw3)  return $is\_inserted;  
[1101:](#3x6dejw)  }  
[1102:](#2cbnorp)   
[1103:](#rgxyzi)  /\*\*  
[1104:](#3bglhnb)  \* Check if the requests to a controller method exceed a limit  
[1105:](#1qlvrv4)  \*  
[1106:](#4aljaix)  \* @access protected  
[1107:](#2pqtkqq)  \* @param string $controller\_method The method being called  
[1108:](#14w3uyj)  \* @return bool TRUE the call limit is below the threshold; otherwise, FALSE  
[1109:](#3ovrdmc)  \*/  
[1110:](#2411nu5)  protected function \_check\_limit($controller\_method)  
[1111:](#j6by1y)  {  
[1112:](#335zgpr)  // They are special, or it might not even have a limit  
[1113:](#1ib9qxk)  if (empty($this->rest->ignore\_limits) === FALSE)  
[1114:](#42ax9ld)  {  
[1115:](#2hg7jt6)  // Everything is fine  
[1116:](#wlhu0z)  return TRUE;  
[1117:](#3gl5cos)  }  
[1118:](#1vqfmwl)   
[1119:](#4fq35ke)  switch ($this->config->item('rest\_limits\_method'))  
[1120:](#2uvdfs7)  {  
[1121:](#1a0nq00)  case 'API\_KEY':  
[1122:](#3u0b8nt)  $limited\_uri = 'api-key:' . (isset($this->rest->key) ? $this->rest->key : '');  
[1123:](#295livm)  $limited\_method\_name = isset($this->rest->key) ? $this->rest->key : '';  
[1124:](#oavt3f)  break;  
[1125:](#38ajbr8)   
[1126:](#1nftlz1)  case 'METHOD\_NAME':  
[1127:](#47fh4mu)  $limited\_uri = 'method-name:' . $controller\_method;  
[1128:](#2mkreun)  $limited\_method\_name = $controller\_method;  
[1129:](#11q1p2g)  break;  
[1130:](#3lpp7q9)   
[1131:](#20uzhy2)  case 'ROUTED\_URL':  
[1132:](#4kun0lv)  default:  
[1133:](#2zzxato)  $limited\_uri = $this->uri->ruri\_string();  
[1134:](#1f57l1h)  if (strpos(strrev($limited\_uri), strrev($this->response->format)) === 0)  
[1135:](#3z4v3pa)  {  
[1136:](#2ea5dx3)  $limited\_uri = substr($limited\_uri,0, -strlen($this->response->format) - 1);  
[1137:](#tffo4w)  }  
[1138:](#3df36sp)  $limited\_uri = 'uri:'.$limited\_uri.':'.$this->request->method; // It's good to differentiate GET from PUT  
[1139:](#1skdh0i)  $limited\_method\_name = $controller\_method;  
[1140:](#4ck0zob)  break;  
[1141:](#2rpb9w4)  }  
[1142:](#16ulk3x)   
[1143:](#3qu92rq)  if (isset($this->methods[$limited\_method\_name]['limit']) === FALSE )  
[1144:](#25zjczj)  {  
[1145:](#l4tn7c)  // Everything is fine  
[1146:](#354h5v5)  return TRUE;  
[1147:](#1k9rg2y)  }  
[1148:](#449eyqr)   
[1149:](#2jep8yk)  // How many times can you get to this method in a defined time\_limit (default: 1 hour)?  
[1150:](#yjzj6d)  $limit = $this->methods[$limited\_method\_name]['limit'];  
[1151:](#3ijn1u6)   
[1152:](#1xoxc1z)  $time\_limit = (isset($this->methods[$limited\_method\_name]['time']) ? $this->methods[$limited\_method\_name]['time'] : 3600); // 3600 = 60 \* 60  
[1153:](#4hokups)   
[1154:](#2wtv4xl)  // Get data about a keys' usage and limit to one row  
[1155:](#1bz5f5e)  $result = $this->rest->db  
[1156:](#3vysxt7)  ->where('uri', $limited\_uri)  
[1157:](#2b43810)  ->where('api\_key', $this->rest->key)  
[1158:](#q9di8t)  ->get($this->config->item('rest\_limits\_table'))  
[1159:](#3a910wm)  ->row();  
[1160:](#1pebb4f)   
[1161:](#49dyts8)  // No calls have been made for this key  
[1162:](#2oj9401)  if ($result === NULL)  
[1163:](#13oje7u)  {  
[1164:](#3no6wvn)  // Create a new row for the following key  
[1165:](#22th73g)  $this->rest->db->insert($this->config->item('rest\_limits\_table'), [  
[1166:](#hyrhb9)  'uri' => $limited\_uri,  
[1167:](#31yezz2)  'api\_key' => isset($this->rest->key) ? $this->rest->key : '',  
[1168:](#1h3pa6v)  'count' => 1,  
[1169:](#413csuo)  'hour\_started' => time()  
[1170:](#2g8n32h)  ]);  
[1171:](#vdxdaa)  }  
[1172:](#3fdkvy3)   
[1173:](#1uiv65w)  // Been a time limit (or by default an hour) since they called  
[1174:](#4eiiotp)  elseif ($result->hour\_started < (time() - $time\_limit))  
[1175:](#2tnsz1i)  {  
[1176:](#18t399b)  // Reset the started period and count  
[1177:](#3ssqrx4)  $this->rest->db  
[1178:](#27y124x)  ->where('uri', $limited\_uri)  
[1179:](#n3bccq)  ->where('api\_key', isset($this->rest->key) ? $this->rest->key : '')  
[1180:](#372yv0j)  ->set('hour\_started', time())  
[1181:](#1m8958c)  ->set('count', 1)  
[1182:](#467wnw5)  ->update($this->config->item('rest\_limits\_table'));  
[1183:](#2ld6y3y)  }  
[1184:](#10ih8br)   
[1185:](#3ki4qzk)  // They have called within the hour, so lets update  
[1186:](#1znf17d)  else  
[1187:](#4jn2jv6)  {  
[1188:](#2yscu2z)  // The limit has been exceeded  
[1189:](#1dxn4as)  if ($result->count >= $limit)  
[1190:](#3xxamyl)  {  
[1191:](#2d2kx6e)  return FALSE;  
[1192:](#s7v7e7)  }  
[1193:](#3c7iq20)   
[1194:](#1rct09t)  // Increase the count by one  
[1195:](#4bcgixm)  $this->rest->db  
[1196:](#2qhqt5f)  ->where('uri', $limited\_uri)  
[1197:](#15n13d8)  ->where('api\_key', $this->rest->key)  
[1198:](#3pmom11)  ->set('count', 'count + 1', FALSE)  
[1199:](#24ryw8u)  ->update($this->config->item('rest\_limits\_table'));  
[1200:](#jx96gn)  }  
[1201:](#33wwp4g)   
[1202:](#1j26zc9)  return TRUE;  
[1203:](#431ui02)  }  
[1204:](#2i74s7v)   
[1205:](#xcf2fo)  /\*\*  
[1206:](#3hc2l3h)  \* Check if there is a specific auth type set for the current class/method/HTTP-method being called  
[1207:](#1whcvba)  \*  
[1208:](#4gh0dz3)  \* @access protected  
[1209:](#2vmao6w)  \* @return bool  
[1210:](#1arkyep)  \*/  
[1211:](#3ur8h2i)  protected function \_auth\_override\_check()  
[1212:](#29wirab)  {  
[1213:](#p1t1i4)  // Assign the class/method auth type override array from the config  
[1214:](#391gk5x)  $auth\_override\_class\_method = $this->config->item('auth\_override\_class\_method');  
[1215:](#1o6qudq)   
[1216:](#486ed1j)  // Check to see if the override array is even populated  
[1217:](#2nbon9c)  if ( ! empty($auth\_override\_class\_method))  
[1218:](#12gyxh5)  {  
[1219:](#3mgmg4y)  // Check for wildcard flag for rules for classes  
[1220:](#21lwqcr)  if ( ! empty($auth\_override\_class\_method[$this->router->class]['\*'])) // Check for class overrides  
[1221:](#gr70kk)  {  
[1222:](#30quj8d)  // No auth override found, prepare nothing but send back a TRUE override flag  
[1223:](#1fw4tg6)  if ($auth\_override\_class\_method[$this->router->class]['\*'] === 'none')  
[1224:](#3zvsc3z)  {  
[1225:](#2f12mbs)  return TRUE;  
[1226:](#u6cwjl)  }  
[1227:](#3e60f7e)   
[1228:](#1tbapf7)  // Basic auth override found, prepare basic  
[1229:](#4day830)  if ($auth\_override\_class\_method[$this->router->class]['\*'] === 'basic')  
[1230:](#2sg8iat)  {  
[1231:](#17lisim)  $this->\_prepare\_basic\_auth();  
[1232:](#3rl6b6f)   
[1233:](#26qgle8)  return TRUE;  
[1234:](#lvqvm1)  }  
[1235:](#35vee9u)   
[1236:](#1l0oohn)  // Digest auth override found, prepare digest  
[1237:](#450c75g)  if ($auth\_override\_class\_method[$this->router->class]['\*'] === 'digest')  
[1238:](#2k5mhd9)  {  
[1239:](#zawrl2)  $this->\_prepare\_digest\_auth();  
[1240:](#3jaka8v)   
[1241:](#1yfukgo)  return TRUE;  
[1242:](#4ifi34h)  }  
[1243:](#2xksdca)   
[1244:](#1cq2nk3)  // Session auth override found, check session  
[1245:](#3wpq67w)  if ($auth\_override\_class\_method[$this->router->class]['\*'] === 'session')  
[1246:](#2bv0gfp)  {  
[1247:](#r0aqni)  $this->\_check\_php\_session();  
[1248:](#3azy9bb)   
[1249:](#1q58jj4)  return TRUE;  
[1250:](#4a4w26x)  }  
[1251:](#2pa6ceq)   
[1252:](#14fgmmj)  // Whitelist auth override found, check client's ip against config whitelist  
[1253:](#3of45ac)  if ($auth\_override\_class\_method[$this->router->class]['\*'] === 'whitelist')  
[1254:](#23kefi5)  {  
[1255:](#ipoppy)  $this->\_check\_whitelist\_auth();  
[1256:](#32pc8dr)   
[1257:](#1humilk)  return TRUE;  
[1258:](#41ua19d)  }  
[1259:](#2gzkbh6)  }  
[1260:](#w4uloz)   
[1261:](#3g4i4cs)  // Check to see if there's an override value set for the current class/method being called  
[1262:](#1v9sekl)  if ( ! empty($auth\_override\_class\_method[$this->router->class][$this->router->method]))  
[1263:](#4f9fx8e)  {  
[1264:](#2ueq7g7)  // None auth override found, prepare nothing but send back a TRUE override flag  
[1265:](#19k0ho0)  if ($auth\_override\_class\_method[$this->router->class][$this->router->method] === 'none')  
[1266:](#3tjo0bt)  {  
[1267:](#28oyajm)  return TRUE;  
[1268:](#nu8krf)  }  
[1269:](#37tw3f8)   
[1270:](#1mz6dn1)  // Basic auth override found, prepare basic  
[1271:](#46ytwau)  if ($auth\_override\_class\_method[$this->router->class][$this->router->method] === 'basic')  
[1272:](#2m446in)  {  
[1273:](#119egqg)  $this->\_prepare\_basic\_auth();  
[1274:](#3l91ze9)   
[1275:](#20ec9m2)  return TRUE;  
[1276:](#4kdzs9v)  }  
[1277:](#2zja2ho)   
[1278:](#1eokcph)  // Digest auth override found, prepare digest  
[1279:](#3yo7vda)  if ($auth\_override\_class\_method[$this->router->class][$this->router->method] === 'digest')  
[1280:](#2dti5l3)  {  
[1281:](#sysfsw)  $this->\_prepare\_digest\_auth();  
[1282:](#3cyfygp)   
[1283:](#1s3q8oi)  return TRUE;  
[1284:](#4c3drcb)  }  
[1285:](#2r8o1k4)   
[1286:](#16dybrx)  // Session auth override found, check session  
[1287:](#3qdlufq)  if ($auth\_override\_class\_method[$this->router->class][$this->router->method] === 'session')  
[1288:](#25iw4nj)  {  
[1289:](#ko6evc)  $this->\_check\_php\_session();  
[1290:](#34ntxj5)   
[1291:](#1jt47qy)  return TRUE;  
[1292:](#43srqer)  }  
[1293:](#2iy20mk)   
[1294:](#y3caud)  // Whitelist auth override found, check client's ip against config whitelist  
[1295:](#3i2zti6)  if ($auth\_override\_class\_method[$this->router->class][$this->router->method] === 'whitelist')  
[1296:](#1x8a3pz)  {  
[1297:](#4h7xmds)  $this->\_check\_whitelist\_auth();  
[1298:](#2wd7wll)   
[1299:](#1bii6te)  return TRUE;  
[1300:](#3vi5ph7)  }  
[1301:](#2anfzp0)  }  
[1302:](#psq9wt)  }  
[1303:](#39sdskm)   
[1304:](#1oxo2sf)  // Assign the class/method/HTTP-method auth type override array from the config  
[1305:](#48xblg8)  $auth\_override\_class\_method\_http = $this->config->item('auth\_override\_class\_method\_http');  
[1306:](#2o2lvo1)   
[1307:](#137w5vu)  // Check to see if the override array is even populated  
[1308:](#3n7jojn)  if ( ! empty($auth\_override\_class\_method\_http))  
[1309:](#22ctyrg)  {  
[1310:](#hi48z9)  // check for wildcard flag for rules for classes  
[1311:](#31hrrn2)  if ( ! empty($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method]))  
[1312:](#1gn21uv)  {  
[1313:](#40mpkio)  // None auth override found, prepare nothing but send back a TRUE override flag  
[1314:](#2frzuqh)  if ($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method] === 'none')  
[1315:](#uxa4ya)  {  
[1316:](#3ewxnm3)  return TRUE;  
[1317:](#1u27xtw)  }  
[1318:](#4e1vghp)   
[1319:](#2t75qpi)  // Basic auth override found, prepare basic  
[1320:](#18cg0xb)  if ($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method] === 'basic')  
[1321:](#3sc3jl4)  {  
[1322:](#27hdtsx)  $this->\_prepare\_basic\_auth();  
[1323:](#mmo40q)   
[1324:](#36mbmoj)  return TRUE;  
[1325:](#1lrlwwc)  }  
[1326:](#45r9fk5)   
[1327:](#2kwjpry)  // Digest auth override found, prepare digest  
[1328:](#101tzzr)  if ($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method] === 'digest')  
[1329:](#3k1hink)  {  
[1330:](#1z6rsvd)  $this->\_prepare\_digest\_auth();  
[1331:](#4j6fbj6)   
[1332:](#2ybplqz)  return TRUE;  
[1333:](#1dgzvys)  }  
[1334:](#3xgneml)   
[1335:](#2clxoue)  // Session auth override found, check session  
[1336:](#rr7z27)  if ($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method] === 'session')  
[1337:](#3bqvhq0)  {  
[1338:](#1qw5rxt)  $this->\_check\_php\_session();  
[1339:](#4avtalm)   
[1340:](#2q13ktf)  return TRUE;  
[1341:](#156dv18)  }  
[1342:](#3p61dp1)   
[1343:](#24bbnwu)  // Whitelist auth override found, check client's ip against config whitelist  
[1344:](#jgly4n)  if ($auth\_override\_class\_method\_http[$this->router->class]['\*'][$this->request->method] === 'whitelist')  
[1345:](#33g9gsg)  {  
[1346:](#1iljr09)  $this->\_check\_whitelist\_auth();  
[1347:](#42l79o2)   
[1348:](#2hqhjvv)  return TRUE;  
[1349:](#wvru3o)  }  
[1350:](#3gvfcrh)  }  
[1351:](#1w0pmza)   
[1352:](#4g0d5n3)  // Check to see if there's an override value set for the current class/method/HTTP-method being called  
[1353:](#2v5nfuw)  if ( ! empty($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method]))  
[1354:](#1aaxq2p)  {  
[1355:](#3ual8qi)  // None auth override found, prepare nothing but send back a TRUE override flag  
[1356:](#29fviyb)  if ($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method] === 'none')  
[1357:](#ol5t64)  {  
[1358:](#38ktbtx)  return TRUE;  
[1359:](#1nq3m1q)  }  
[1360:](#47pr4pj)   
[1361:](#2mv1exc)  // Basic auth override found, prepare basic  
[1362:](#120bp55)  if ($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method] === 'basic')  
[1363:](#3lzz7sy)  {  
[1364:](#2159i0r)  $this->\_prepare\_basic\_auth();  
[1365:](#4l4x0ok)   
[1366:](#30a7awd)  return TRUE;  
[1367:](#1ffhl46)  }  
[1368:](#3zf53rz)   
[1369:](#2ekfdzs)  // Digest auth override found, prepare digest  
[1370:](#tppo7l)  if ($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method] === 'digest')  
[1371:](#3dpd6ve)  {  
[1372:](#1sunh37)  $this->\_prepare\_digest\_auth();  
[1373:](#4cuazr0)   
[1374:](#2rzl9yt)  return TRUE;  
[1375:](#174vk6m)  }  
[1376:](#3r4j2uf)   
[1377:](#269td28)  // Session auth override found, check session  
[1378:](#lf3na1)  if ($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method] === 'session')  
[1379:](#35er5xu)  {  
[1380:](#1kk1g5n)  $this->\_check\_php\_session();  
[1381:](#44joytg)   
[1382:](#2joz919)  return TRUE;  
[1383:](#yu9j92)  }  
[1384:](#3itx1wv)   
[1385:](#1xz7c4o)  // Whitelist auth override found, check client's ip against config whitelist  
[1386:](#4hyuush)  if ($auth\_override\_class\_method\_http[$this->router->class][$this->router->method][$this->request->method] === 'whitelist')  
[1387:](#2x4550a)  {  
[1388:](#1c9ff83)  $this->\_check\_whitelist\_auth();  
[1389:](#3w92xvw)   
[1390:](#2bed83p)  return TRUE;  
[1391:](#qjnibi)  }  
[1392:](#3ajb0zb)  }  
[1393:](#1polb74)  }  
[1394:](#49o8tux)  return FALSE;  
[1395:](#2otj42q)  }  
[1396:](#13yteaj)   
[1397:](#3nygwyc)  /\*\*  
[1398:](#233r765)  \* Parse the GET request arguments  
[1399:](#i91hdy)  \*  
[1400:](#328p01r)  \* @access protected  
[1401:](#1hdza9k)  \* @return void  
[1402:](#41dmsxd)  \*/  
[1403:](#2gix356)  protected function \_parse\_get()  
[1404:](#vo7dcz)  {  
[1405:](#3fnuw0s)  // Merge both the URI segments and query parameters  
[1406:](#1ut568l)  $this->\_get\_args = array\_merge($this->\_get\_args, $this->\_query\_args);  
[1407:](#4essowe)  }  
[1408:](#2ty2z47)   
[1409:](#193d9c0)  /\*\*  
[1410:](#3t30rzt)  \* Parse the POST request arguments  
[1411:](#288b27m)  \*  
[1412:](#ndlcff)  \* @access protected  
[1413:](#37d8v38)  \* @return void  
[1414:](#1mij5b1)  \*/  
[1415:](#46i6nyu)  protected function \_parse\_post()  
[1416:](#2lngy6n)  {  
[1417:](#10sr8eg)  $this->\_post\_args = $\_POST;  
[1418:](#3kser29)   
[1419:](#1zxp1a2)  if ($this->request->format)  
[1420:](#4jxcjxv)  {  
[1421:](#2z2mu5o)  $this->request->body = $this->input->raw\_input\_stream;  
[1422:](#1e7x4dh)  }  
[1423:](#3y7kn1a)  }  
[1424:](#2dcux93)   
[1425:](#si57gw)  /\*\*  
[1426:](#3chsq4p)  \* Parse the PUT request arguments  
[1427:](#1rn30ci)  \*  
[1428:](#4bmqj0b)  \* @access protected  
[1429:](#2qs0t84)  \* @return void  
[1430:](#15xb3fx)  \*/  
[1431:](#3pwym3q)  protected function \_parse\_put()  
[1432:](#2528wbj)  {  
[1433:](#k7j6jc)  if ($this->request->format)  
[1434:](#3476p75)  {  
[1435:](#1jcgzey)  $this->request->body = $this->input->raw\_input\_stream;  
[1436:](#43c4i2r)  }  
[1437:](#2ihesak)  else if ($this->input->method() === 'put')  
[1438:](#xmp2id)  {  
[1439:](#3hmcl66)  // If no filetype is provided, then there are probably just arguments  
[1440:](#1wrmvdz)  $this->\_put\_args = $this->input->input\_stream();  
[1441:](#4grae1s)  }  
[1442:](#2vwko9l)  }  
[1443:](#1b1uyhe)   
[1444:](#3v1ih57)  /\*\*  
[1445:](#2a6srd0)  \* Parse the HEAD request arguments  
[1446:](#pc31kt)  \*  
[1447:](#39bqk8m)  \* @access protected  
[1448:](#1oh0ugf)  \* @return void  
[1449:](#48god48)  \*/  
[1450:](#2nlync1)  protected function \_parse\_head()  
[1451:](#12r8xju)  {  
[1452:](#3mqwg7n)  // Parse the HEAD variables  
[1453:](#21w6qfg)  parse\_str(parse\_url($this->input->server('REQUEST\_URI'), PHP\_URL\_QUERY), $head);  
[1454:](#h1h0n9)   
[1455:](#3114jb2)  // Merge both the URI segments and HEAD params  
[1456:](#1g6etiv)  $this->\_head\_args = array\_merge($this->\_head\_args, $head);  
[1457:](#4062c6o)  }  
[1458:](#2fbcmeh)   
[1459:](#ugmwma)  /\*\*  
[1460:](#3egafa3)  \* Parse the OPTIONS request arguments  
[1461:](#1tlkphw)  \*  
[1462:](#4dl885p)  \* @access protected  
[1463:](#2sqiidi)  \* @return void  
[1464:](#17vsslb)  \*/  
[1465:](#3rvgb94)  protected function \_parse\_options()  
[1466:](#270qlgx)  {  
[1467:](#m60voq)  // Parse the OPTIONS variables  
[1468:](#365oecj)  parse\_str(parse\_url($this->input->server('REQUEST\_URI'), PHP\_URL\_QUERY), $options);  
[1469:](#1layokc)   
[1470:](#45am785)  // Merge both the URI segments and OPTIONS params  
[1471:](#2kfwhfy)  $this->\_options\_args = array\_merge($this->\_options\_args, $options);  
[1472:](#zl6rnr)  }  
[1473:](#3jkuabk)   
[1474:](#1yq4kjd)  /\*\*  
[1475:](#4ips376)  \* Parse the PATCH request arguments  
[1476:](#2xv2dez)  \*  
[1477:](#1d0cnms)  \* @access protected  
[1478:](#3x006al)  \* @return void  
[1479:](#2c5agie)  \*/  
[1480:](#rakqq7)  protected function \_parse\_patch()  
[1481:](#3ba89e0)  {  
[1482:](#1qfijlt)  // It might be a HTTP body  
[1483:](#4af629m)  if ($this->request->format)  
[1484:](#2pkgchf)  {  
[1485:](#14pqmp8)  $this->request->body = $this->input->raw\_input\_stream;  
[1486:](#3ope5d1)  }  
[1487:](#23uofku)  else if ($this->input->method() === 'patch')  
[1488:](#izypsn)  {  
[1489:](#32zm8gg)  // If no filetype is provided, then there are probably just arguments  
[1490:](#1i4wio9)  $this->\_patch\_args = $this->input->input\_stream();  
[1491:](#424k1c2)  }  
[1492:](#2h9ubjv)  }  
[1493:](#wf4lro)   
[1494:](#3ges4fh)  /\*\*  
[1495:](#1vk2ena)  \* Parse the DELETE request arguments  
[1496:](#4fjpxb3)  \*  
[1497:](#2up07iw)  \* @access protected  
[1498:](#19uahqp)  \* @return void  
[1499:](#3tty0ei)  \*/  
[1500:](#28z8amb)  protected function \_parse\_delete()  
[1501:](#o4iku4)  {  
[1502:](#38463hx)  // These should exist if a DELETE request  
[1503:](#1n9gdpq)  if ($this->input->method() === 'delete')  
[1504:](#4793wdj)  {  
[1505:](#2mee6lc)  $this->\_delete\_args = $this->input->input\_stream();  
[1506:](#11jogt5)  }  
[1507:](#3ljbzgy)  }  
[1508:](#20om9or)   
[1509:](#4ko9sck)  /\*\*  
[1510:](#2ztk2kd)  \* Parse the query parameters  
[1511:](#1eyucs6)  \*  
[1512:](#3yyhvfz)  \* @access protected  
[1513:](#2e3s5ns)  \* @return void  
[1514:](#t92fvl)  \*/  
[1515:](#3d8pyje)  protected function \_parse\_query()  
[1516:](#1se08r7)  {  
[1517:](#4cdnrf0)  $this->\_query\_args = $this->input->get();  
[1518:](#2riy1mt)  }  
[1519:](#16o8bum)   
[1520:](#3qnvuif)  // INPUT FUNCTION --------------------------------------------------------------  
[1521:](#25t64q8)   
[1522:](#kygey1)  /\*\*  
[1523:](#34y3xlu)  \* Retrieve a value from a GET request  
[1524:](#1k3e7tn)  \*  
[1525:](#4431qhg)  \* @access public  
[1526:](#2j8c0p9)  \* @param NULL $key Key to retrieve from the GET request  
[1527:](#ydmax2)  \* If NULL an array of arguments is returned  
[1528:](#3id9tkv)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1529:](#1xik3so)  \* @return array|string|NULL Value from the GET request; otherwise, NULL  
[1530:](#4hi7mgh)  \*/  
[1531:](#2wnhwoa)  public function get($key = NULL, $xss\_clean = NULL)  
[1532:](#1bss6w3)  {  
[1533:](#3vsfpjw)  if ($key === NULL)  
[1534:](#2axpzrp)  {  
[1535:](#q309zi)  return $this->\_get\_args;  
[1536:](#3a2nsnb)  }  
[1537:](#1p7y2v4)   
[1538:](#497llix)  return isset($this->\_get\_args[$key]) ? $this->\_xss\_clean($this->\_get\_args[$key], $xss\_clean) : NULL;  
[1539:](#2ocvvqq)  }  
[1540:](#13i65yj)   
[1541:](#3nhtomc)  /\*\*  
[1542:](#22n3yu5)  \* Retrieve a value from a OPTIONS request  
[1543:](#hse91y)  \*  
[1544:](#31s1rpr)  \* @access public  
[1545:](#1gxc1xk)  \* @param NULL $key Key to retrieve from the OPTIONS request.  
[1546:](#40wzkld)  \* If NULL an array of arguments is returned  
[1547:](#2g29ut6)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1548:](#v7k50z)  \* @return array|string|NULL Value from the OPTIONS request; otherwise, NULL  
[1549:](#3f77nos)  \*/  
[1550:](#1uchxwl)  public function options($key = NULL, $xss\_clean = NULL)  
[1551:](#4ec5gke)  {  
[1552:](#2thfqs7)  if ($key === NULL)  
[1553:](#18mq100)  {  
[1554:](#3smdjnt)  return $this->\_options\_args;  
[1555:](#27rntvm)  }  
[1556:](#mwy43f)   
[1557:](#36wlmr8)  return isset($this->\_options\_args[$key]) ? $this->\_xss\_clean($this->\_options\_args[$key], $xss\_clean) : NULL;  
[1558:](#1m1vwz1)  }  
[1559:](#461jfmu)   
[1560:](#2l6tpun)  /\*\*  
[1561:](#10c402g)  \* Retrieve a value from a HEAD request  
[1562:](#3kbriq9)  \*  
[1563:](#1zh1sy2)  \* @access public  
[1564:](#4jgpblv)  \* @param NULL $key Key to retrieve from the HEAD request  
[1565:](#2ylzlto)  \* If NULL an array of arguments is returned  
[1566:](#1dr9w1h)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1567:](#3xqxepa)  \* @return array|string|NULL Value from the HEAD request; otherwise, NULL  
[1568:](#2cw7ox3)  \*/  
[1569:](#s1hz4w)  public function head($key = NULL, $xss\_clean = NULL)  
[1570:](#3c15hsp)  {  
[1571:](#1r6fs0i)  if ($key === NULL)  
[1572:](#4b63aob)  {  
[1573:](#2qbdkw4)  return $this->\_head\_args;  
[1574:](#15gnv3x)  }  
[1575:](#3pgbdrq)   
[1576:](#24llnzj)  return isset($this->\_head\_args[$key]) ? $this->\_xss\_clean($this->\_head\_args[$key], $xss\_clean) : NULL;  
[1577:](#jqvy7c)  }  
[1578:](#33qjgv5)   
[1579:](#1ivtr2y)  /\*\*  
[1580:](#42vh9qr)  \* Retrieve a value from a POST request  
[1581:](#2i0rjyk)  \*  
[1582:](#x61u6d)  \* @access public  
[1583:](#3h5pcu6)  \* @param NULL $key Key to retrieve from the POST request  
[1584:](#1wazn1z)  \* If NULL an array of arguments is returned  
[1585:](#4gan5ps)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1586:](#2vfxfxl)  \* @return array|string|NULL Value from the POST request; otherwise, NULL  
[1587:](#1al7q5e)  \*/  
[1588:](#3ukv8t7)  public function post($key = NULL, $xss\_clean = NULL)  
[1589:](#29q5j10)  {  
[1590:](#ovft8t)  if ($key === NULL)  
[1591:](#38v3bwm)  {  
[1592:](#1o0dm4f)  return $this->\_post\_args;  
[1593:](#48014s8)  }  
[1594:](#2n5bf01)   
[1595:](#12alp7u)  return isset($this->\_post\_args[$key]) ? $this->\_xss\_clean($this->\_post\_args[$key], $xss\_clean) : NULL;  
[1596:](#3ma97vn)  }  
[1597:](#21fji3g)   
[1598:](#gktsb9)  /\*\*  
[1599:](#30khaz2)  \* Retrieve a value from a PUT request  
[1600:](#1fprl6v)  \*  
[1601:](#3zpf3uo)  \* @access public  
[1602:](#2eupe2h)  \* @param NULL $key Key to retrieve from the PUT request  
[1603:](#tzzoaa)  \* If NULL an array of arguments is returned  
[1604:](#3dzn6y3)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1605:](#1t4xh5w)  \* @return array|string|NULL Value from the PUT request; otherwise, NULL  
[1606:](#4d4kztp)  \*/  
[1607:](#2s9va1i)  public function put($key = NULL, $xss\_clean = NULL)  
[1608:](#17f5k9b)  {  
[1609:](#3ret2x4)  if ($key === NULL)  
[1610:](#26k3d4x)  {  
[1611:](#lpdncq)  return $this->\_put\_args;  
[1612:](#35p160j)  }  
[1613:](#1kubg8c)   
[1614:](#44tyyw5)  return isset($this->\_put\_args[$key]) ? $this->\_xss\_clean($this->\_put\_args[$key], $xss\_clean) : NULL;  
[1615:](#2jz993y)  }  
[1616:](#z4jjbr)   
[1617:](#3j471zk)  /\*\*  
[1618:](#1y9hc7d)  \* Retrieve a value from a DELETE request  
[1619:](#4i94uv6)  \*  
[1620:](#2xef52z)  \* @access public  
[1621:](#1cjpfas)  \* @param NULL $key Key to retrieve from the DELETE request  
[1622:](#3wjcxyl)  \* If NULL an array of arguments is returned  
[1623:](#2bon86e)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1624:](#qtxie7)  \* @return array|string|NULL Value from the DELETE request; otherwise, NULL  
[1625:](#3atl120)  \*/  
[1626:](#1pyvb9t)  public function delete($key = NULL, $xss\_clean = NULL)  
[1627:](#49yitxm)  {  
[1628:](#2p3t45f)  if ($key === NULL)  
[1629:](#1493ed8)  {  
[1630:](#3o8qx11)  return $this->\_delete\_args;  
[1631:](#23e178u)  }  
[1632:](#ijbhgn)   
[1633:](#32iz04g)  return isset($this->\_delete\_args[$key]) ? $this->\_xss\_clean($this->\_delete\_args[$key], $xss\_clean) : NULL;  
[1634:](#1ho9ac9)  }  
[1635:](#41nwt02)   
[1636:](#2gt737v)  /\*\*  
[1637:](#vyhdfo)  \* Retrieve a value from a PATCH request  
[1638:](#3fy4w3h)  \*  
[1639:](#1v3f6ba)  \* @access public  
[1640:](#4f32oz3)  \* @param NULL $key Key to retrieve from the PATCH request  
[1641:](#2u8cz6w)  \* If NULL an array of arguments is returned  
[1642:](#19dn9ep)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1643:](#3tdas2i)  \* @return array|string|NULL Value from the PATCH request; otherwise, NULL  
[1644:](#28il2ab)  \*/  
[1645:](#nnvci4)  public function patch($key = NULL, $xss\_clean = NULL)  
[1646:](#37niv5x)  {  
[1647:](#1mst5dq)  if ($key === NULL)  
[1648:](#46sgo1j)  {  
[1649:](#2lxqy9c)  return $this->\_patch\_args;  
[1650:](#11318h5)  }  
[1651:](#3l2or4y)   
[1652:](#207z1cr)  return isset($this->\_patch\_args[$key]) ? $this->\_xss\_clean($this->\_patch\_args[$key], $xss\_clean) : NULL;  
[1653:](#4k7mk0k)  }  
[1654:](#2zcwu8d)   
[1655:](#1ei74g6)  /\*\*  
[1656:](#3yhun3z)  \* Retrieve a value from the query parameters  
[1657:](#2dn4xbs)  \*  
[1658:](#ssf7jl)  \* @access public  
[1659:](#3cs2q7e)  \* @param NULL $key Key to retrieve from the query parameters  
[1660:](#1rxd0f7)  \* If NULL an array of arguments is returned  
[1661:](#4bx0j30)  \* @param NULL $xss\_clean Whether to apply XSS filtering  
[1662:](#2r2atat)  \* @return array|string|NULL Value from the query parameters; otherwise, NULL  
[1663:](#167l3im)  \*/  
[1664:](#3q78m6f)  public function query($key = NULL, $xss\_clean = NULL)  
[1665:](#25ciwe8)  {  
[1666:](#kht6m1)  if ($key === NULL)  
[1667:](#34hgp9u)  {  
[1668:](#1jmqzhn)  return $this->\_query\_args;  
[1669:](#43mei5g)  }  
[1670:](#2irosd9)   
[1671:](#xwz2l2)  return isset($this->\_query\_args[$key]) ? $this->\_xss\_clean($this->\_query\_args[$key], $xss\_clean) : NULL;  
[1672:](#3hwml8v)  }  
[1673:](#1x1wvgo)   
[1674:](#4h1ke4h)  /\*\*  
[1675:](#2w6uoca)  \* Sanitizes data so that Cross Site Scripting Hacks can be  
[1676:](#1bc4yk3)  \* prevented  
[1677:](#3vbsh7w)  \*  
[1678:](#2ah2rfp)  \* @access protected  
[1679:](#pmd1ni)  \* @param string $value Input data  
[1680:](#39m0kbb)  \* @param bool $xss\_clean Whether to apply XSS filtering  
[1681:](#1orauj4)  \* @return string  
[1682:](#48qyd6x)  \*/  
[1683:](#2nw8neq)  protected function \_xss\_clean($value, $xss\_clean)  
[1684:](#131ixmj)  {  
[1685:](#3n16gac)  is\_bool($xss\_clean) || $xss\_clean = $this->\_enable\_xss;  
[1686:](#226gqi5)   
[1687:](#hbr0py)  return $xss\_clean === TRUE ? $this->security->xss\_clean($value) : $value;  
[1688:](#31bejdr)  }  
[1689:](#1ggotlk)   
[1690:](#40gcc9d)  /\*\*  
[1691:](#2flmmh6)  \* Retrieve the validation errors  
[1692:](#uqwwoz)  \*  
[1693:](#3eqkfcs)  \* @access public  
[1694:](#1tvupkl)  \* @return array  
[1695:](#4dvi88e)  \*/  
[1696:](#2t0sig7)  public function validation\_errors()  
[1697:](#1862so0)  {  
[1698:](#3s5qbbt)  $string = strip\_tags($this->form\_validation->error\_string());  
[1699:](#27b0ljm)   
[1700:](#mgavrf)  return explode(PHP\_EOL, trim($string, PHP\_EOL));  
[1701:](#36fyef8)  }  
[1702:](#1ll8on1)   
[1703:](#45kw7au)  // SECURITY FUNCTIONS ---------------------------------------------------------  
[1704:](#2kq6hin)   
[1705:](#zvgrqg)  /\*\*  
[1706:](#3jv4ae9)  \* Perform LDAP Authentication  
[1707:](#1z0ekm2)  \*  
[1708:](#4j0239v)  \* @access protected  
[1709:](#2y5cdho)  \* @param string $username The username to validate  
[1710:](#1damnph)  \* @param string $password The password to validate  
[1711:](#3xaa6da)  \* @return bool  
[1712:](#2cfkgl3)  \*/  
[1713:](#rkuqsw)  protected function \_perform\_ldap\_auth($username = '', $password = NULL)  
[1714:](#3bki9gp)  {  
[1715:](#1qpsjoi)  if (empty($username))  
[1716:](#4apg2cb)  {  
[1717:](#2puqck4)  log\_message('debug', 'LDAP Auth: failure, empty username');  
[1718:](#1500mrx)  return FALSE;  
[1719:](#3ozo5fq)  }  
[1720:](#244yfnj)   
[1721:](#ja8pvc)  log\_message('debug', 'LDAP Auth: Loading configuration');  
[1722:](#339w8j5)   
[1723:](#1if6iqy)  $this->config->load('ldap.php', TRUE);  
[1724:](#42eu1er)   
[1725:](#2hk4bmk)  $ldap = [  
[1726:](#wpelud)  'timeout' => $this->config->item('timeout', 'ldap'),  
[1727:](#3gp24i6)  'host' => $this->config->item('server', 'ldap'),  
[1728:](#1vucepz)  'port' => $this->config->item('port', 'ldap'),  
[1729:](#4ftzxds)  'rdn' => $this->config->item('binduser', 'ldap'),  
[1730:](#2uza7ll)  'pass' => $this->config->item('bindpw', 'ldap'),  
[1731:](#1a4khte)  'basedn' => $this->config->item('basedn', 'ldap'),  
[1732:](#3u480h7)  ];  
[1733:](#299iap0)   
[1734:](#oeskwt)  log\_message('debug', 'LDAP Auth: Connect to ' . (isset($ldaphost) ? $ldaphost : '[ldap not configured]'));  
[1735:](#38eg3km)   
[1736:](#1njqdsf)  // Connect to the ldap server  
[1737:](#47jdwg8)  $ldapconn = ldap\_connect($ldap['host'], $ldap['port']);  
[1738:](#2moo6o1)  if ($ldapconn)  
[1739:](#11tygvu)  {  
[1740:](#3ltlzjn)  log\_message('debug', 'Setting timeout to '.$ldap['timeout'].' seconds');  
[1741:](#20yw9rg)   
[1742:](#4kyjsf9)  ldap\_set\_option($ldapconn, LDAP\_OPT\_NETWORK\_TIMEOUT, $ldap['timeout']);  
[1743:](#303u2n2)   
[1744:](#1f94cuv)  log\_message('debug', 'LDAP Auth: Binding to '.$ldap['host'].' with dn '.$ldap['rdn']);  
[1745:](#3z8rvio)   
[1746:](#2ee25qh)  // Binding to the ldap server  
[1747:](#tjcfya)  $ldapbind = ldap\_bind($ldapconn, $ldap['rdn'], $ldap['pass']);  
[1748:](#3dizym3)   
[1749:](#1soa8tw)  // Verify the binding  
[1750:](#4cnxrhp)  if ($ldapbind === FALSE)  
[1751:](#2rt81pi)  {  
[1752:](#16yibxb)  log\_message('error', 'LDAP Auth: bind was unsuccessful');  
[1753:](#3qy5ul4)  return FALSE;  
[1754:](#263g4sx)  }  
[1755:](#l8qf0q)   
[1756:](#358dxoj)  log\_message('debug', 'LDAP Auth: bind successful');  
[1757:](#1kdo7wc)  }  
[1758:](#44dbqk5)   
[1759:](#2jim0ry)  // Search for user  
[1760:](#ynwazr)  if (($res\_id = ldap\_search($ldapconn, $ldap['basedn'], "uid=$username")) === FALSE)  
[1761:](#3injtnk)  {  
[1762:](#1xsu3vd)  log\_message('error', 'LDAP Auth: User '.$username.' not found in search');  
[1763:](#4hshmj6)  return FALSE;  
[1764:](#2wxrwqz)  }  
[1765:](#1c326ys)   
[1766:](#3w2ppml)  if (ldap\_count\_entries($ldapconn, $res\_id) !== 1)  
[1767:](#2b7zzue)  {  
[1768:](#qdaa27)  log\_message('error', 'LDAP Auth: Failure, username '.$username.'found more than once');  
[1769:](#3acxsq0)  return FALSE;  
[1770:](#1pi82xt)  }  
[1771:](#49hvllm)   
[1772:](#2on5vtf)  if (($entry\_id = ldap\_first\_entry($ldapconn, $res\_id)) === FALSE)  
[1773:](#13sg618)  {  
[1774:](#3ns3op1)  log\_message('error', 'LDAP Auth: Failure, entry of search result could not be fetched');  
[1775:](#22xdywu)  return FALSE;  
[1776:](#i2o94n)  }  
[1777:](#322brsg)   
[1778:](#1h7m209)  if (($user\_dn = ldap\_get\_dn($ldapconn, $entry\_id)) === FALSE)  
[1779:](#4179ko2)  {  
[1780:](#2gcjuvv)  log\_message('error', 'LDAP Auth: Failure, user-dn could not be fetched');  
[1781:](#vhu53o)  return FALSE;  
[1782:](#3fhhnrh)  }  
[1783:](#1umrxza)   
[1784:](#4emfgn3)  // User found, could not authenticate as user  
[1785:](#2trpquw)  if (($link\_id = ldap\_bind($ldapconn, $user\_dn, $password)) === FALSE)  
[1786:](#18x012p)  {  
[1787:](#3swnjqi)  log\_message('error', 'LDAP Auth: Failure, username/password did not match: ' . $user\_dn);  
[1788:](#281xtyb)  return FALSE;  
[1789:](#n78464)  }  
[1790:](#376vmtx)   
[1791:](#1mc5x1q)  log\_message('debug', 'LDAP Auth: Success '.$user\_dn.' authenticated successfully');  
[1792:](#46btfpj)   
[1793:](#2lh3pxc)  $this->\_user\_ldap\_dn = $user\_dn;  
[1794:](#10me055)   
[1795:](#3km1isy)  ldap\_close($ldapconn);  
[1796:](#1zrbt0r)   
[1797:](#4jqzbok)  return TRUE;  
[1798:](#2yw9lwd)  }  
[1799:](#1e1jw46)   
[1800:](#3y17erz)  /\*\*  
[1801:](#2d6hozs)  \* Perform Library Authentication - Override this function to change the way the library is called  
[1802:](#sbrz7l)  \*  
[1803:](#3cbfhve)  \* @access protected  
[1804:](#1rgps37)  \* @param string $username The username to validate  
[1805:](#4bgdar0)  \* @param string $password The password to validate  
[1806:](#2qlnkyt)  \* @return bool  
[1807:](#15qxv6m)  \*/  
[1808:](#3pqlduf)  protected function \_perform\_library\_auth($username = '', $password = NULL)  
[1809:](#24vvo28)  {  
[1810:](#k15ya1)  if (empty($username))  
[1811:](#340tgxu)  {  
[1812:](#1j63r5n)  log\_message('error', 'Library Auth: Failure, empty username');  
[1813:](#435r9tg)  return FALSE;  
[1814:](#2ib1k19)  }  
[1815:](#xgbu92)   
[1816:](#3hfzcwv)  $auth\_library\_class = strtolower($this->config->item('auth\_library\_class'));  
[1817:](#1wl9n4o)  $auth\_library\_function = strtolower($this->config->item('auth\_library\_function'));  
[1818:](#4gkx5sh)   
[1819:](#2vq7g0a)  if (empty($auth\_library\_class))  
[1820:](#1avhq83)  {  
[1821:](#3uv58vw)  log\_message('debug', 'Library Auth: Failure, empty auth\_library\_class');  
[1822:](#2a0fj3p)  return FALSE;  
[1823:](#p5ptbi)  }  
[1824:](#395dbzb)   
[1825:](#1oanm74)  if (empty($auth\_library\_function))  
[1826:](#48ab4ux)  {  
[1827:](#2nflf2q)  log\_message('debug', 'Library Auth: Failure, empty auth\_library\_function');  
[1828:](#12kvpaj)  return FALSE;  
[1829:](#3mkj7yc)  }  
[1830:](#21pti65)   
[1831:](#gv3sdy)  if (is\_callable([$auth\_library\_class, $auth\_library\_function]) === FALSE)  
[1832:](#30urb1r)  {  
[1833:](#1g01l9k)  $this->load->library($auth\_library\_class);  
[1834:](#3zzp3xd)  }  
[1835:](#2f4ze56)   
[1836:](#ua9ocz)  return $this->{$auth\_library\_class}->$auth\_library\_function($username, $password);  
[1837:](#3e9x70s)  }  
[1838:](#1tf7h8l)   
[1839:](#4deuzwe)  /\*\*  
[1840:](#2sk5a47)  \* Check if the user is logged in  
[1841:](#17pfkc0)  \*  
[1842:](#3rp32zt)  \* @access protected  
[1843:](#26udd7m)  \* @param string $username The user's name  
[1844:](#lznnff)  \* @param bool|string $password The user's password  
[1845:](#35zb638)  \* @return bool  
[1846:](#1l4lgb1)  \*/  
[1847:](#4548yyu)  protected function \_check\_login($username = NULL, $password = FALSE)  
[1848:](#2k9j96n)  {  
[1849:](#zetjeg)  if (empty($username))  
[1850:](#3jeh229)  {  
[1851:](#1yjrca2)  return FALSE;  
[1852:](#4ijeuxv)  }  
[1853:](#2xop55o)   
[1854:](#1ctzfdh)  $auth\_source = strtolower($this->config->item('auth\_source'));  
[1855:](#3wtmy1a)  $rest\_auth = strtolower($this->config->item('rest\_auth'));  
[1856:](#2byx893)  $valid\_logins = $this->config->item('rest\_valid\_logins');  
[1857:](#r47igw)   
[1858:](#3b3v14p)  if ( ! $this->config->item('auth\_source') && $rest\_auth === 'digest')  
[1859:](#1q95bci)  {  
[1860:](#4a8su0b)  // For digest we do not have a password passed as argument  
[1861:](#2pe3484)  return md5($username.':'.$this->config->item('rest\_realm').':'.(isset($valid\_logins[$username]) ? $valid\_logins[$username] : ''));  
[1862:](#14jdefx)  }  
[1863:](#3oj0x3q)   
[1864:](#23ob7bj)  if ($password === FALSE)  
[1865:](#itlhjc)  {  
[1866:](#32t9075)  return FALSE;  
[1867:](#1hyjaey)  }  
[1868:](#41y6t2r)   
[1869:](#2h3h3ak)  if ($auth\_source === 'ldap')  
[1870:](#w8rdid)  {  
[1871:](#3g8ew66)  log\_message('debug', "Performing LDAP authentication for $username");  
[1872:](#1vdp6dz)   
[1873:](#4fdcp1s)  return $this->\_perform\_ldap\_auth($username, $password);  
[1874:](#2uimz9l)  }  
[1875:](#19nx9he)   
[1876:](#3tnks57)  if ($auth\_source === 'library')  
[1877:](#28sv2d0)  {  
[1878:](#ny5ckt)  log\_message('debug', "Performing Library authentication for $username");  
[1879:](#37xsv8m)   
[1880:](#1n335gf)  return $this->\_perform\_library\_auth($username, $password);  
[1881:](#472qo48)  }  
[1882:](#2m80yc1)   
[1883:](#11db8ju)  if (array\_key\_exists($username, $valid\_logins) === FALSE)  
[1884:](#3lcyr7n)  {  
[1885:](#20i91fg)  return FALSE;  
[1886:](#4khwk39)  }  
[1887:](#2zn6ub2)   
[1888:](#1esh4iv)  if ($valid\_logins[$username] !== $password)  
[1889:](#3ys4n6o)  {  
[1890:](#2dxexeh)  return FALSE;  
[1891:](#t2p7ma)  }  
[1892:](#3d2cqa3)   
[1893:](#1s7n0hw)  return TRUE;  
[1894:](#4c7aj5p)  }  
[1895:](#2rcktdi)   
[1896:](#16hv3lb)  /\*\*  
[1897:](#3qhim94)  \* Check to see if the user is logged in with a PHP session key  
[1898:](#25mswgx)  \*  
[1899:](#ks36oq)  \* @access protected  
[1900:](#34rqpcj)  \* @return void  
[1901:](#1jx0zkc)  \*/  
[1902:](#43woi85)  protected function \_check\_php\_session()  
[1903:](#2j1ysfy)  {  
[1904:](#y792nr)  // Get the auth\_source config item  
[1905:](#3i6wlbk)  $key = $this->config->item('auth\_source');  
[1906:](#1xc6vjd)   
[1907:](#4hbue76)  // If falsy, then the user isn't logged in  
[1908:](#2wh4oez)  if ( ! $this->session->userdata($key))  
[1909:](#1bmeyms)  {  
[1910:](#3vm2hal)  // Display an error response  
[1911:](#2arcrie)  $this->response([  
[1912:](#pwn1q7)  $this->config->item('rest\_status\_field\_name') => FALSE,  
[1913:](#39wake0)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_unauthorized')  
[1914:](#1p1kult)  ], self::HTTP\_UNAUTHORIZED);  
[1915:](#4918d9m)  }  
[1916:](#2o6inhf)  }  
[1917:](#13bsxp8)   
[1918:](#3nbggd1)  /\*\*  
[1919:](#22gqqku)  \* Prepares for basic authentication  
[1920:](#hm10sn)  \*  
[1921:](#31lojgg)  \* @access protected  
[1922:](#1gqyto9)  \* @return void  
[1923:](#40qmcc2)  \*/  
[1924:](#2fvwmjv)  protected function \_prepare\_basic\_auth()  
[1925:](#v16wro)  {  
[1926:](#3f0uffh)  // If whitelist is enabled it has the first chance to kick them out  
[1927:](#1u64pna)  if ($this->config->item('rest\_ip\_whitelist\_enabled'))  
[1928:](#4e5s8b3)  {  
[1929:](#2tb2iiw)  $this->\_check\_whitelist\_auth();  
[1930:](#18gcsqp)  }  
[1931:](#3sg0bei)   
[1932:](#27lalmb)  // Returns NULL if the SERVER variables PHP\_AUTH\_USER and HTTP\_AUTHENTICATION don't exist  
[1933:](#mqkvu4)  $username = $this->input->server('PHP\_AUTH\_USER');  
[1934:](#36q8ehx)  $http\_auth = $this->input->server('HTTP\_AUTHENTICATION');  
[1935:](#1lviopq)   
[1936:](#45v67dj)  $password = NULL;  
[1937:](#2l0ghlc)  if ($username !== NULL)  
[1938:](#105qrt5)  {  
[1939:](#3k5eagy)  $password = $this->input->server('PHP\_AUTH\_PW');  
[1940:](#1zaokor)  }  
[1941:](#4jac3ck)  elseif ($http\_auth !== NULL)  
[1942:](#2yfmdkd)  {  
[1943:](#1dkwns6)  // If the authentication header is set as basic, then extract the username and password from  
[1944:](#3xkk6fz)  // HTTP\_AUTHORIZATION e.g. my\_username:my\_password. This is passed in the .htaccess file  
[1945:](#2cpugns)  if (strpos(strtolower($http\_auth), 'basic') === 0)  
[1946:](#rv4qvl)  {  
[1947:](#3bus9je)  // Search online for HTTP\_AUTHORIZATION workaround to explain what this is doing  
[1948:](#1r02jr7)  list($username, $password) = explode(':', base64\_decode(substr($this->input->server('HTTP\_AUTHORIZATION'), 6)));  
[1949:](#4azq2f0)  }  
[1950:](#2q50cmt)  }  
[1951:](#15aamum)   
[1952:](#3p9y5if)  // Check if the user is logged into the system  
[1953:](#24f8fq8)  if ($this->\_check\_login($username, $password) === FALSE)  
[1954:](#jkipy1)  {  
[1955:](#33k68lu)  $this->\_force\_login();  
[1956:](#1ipgitn)  }  
[1957:](#42p41hg)  }  
[1958:](#2huebp9)   
[1959:](#wzolx2)  /\*\*  
[1960:](#3gzc4kv)  \* Prepares for digest authentication  
[1961:](#1w4meso)  \*  
[1962:](#4g49xgh)  \* @access protected  
[1963:](#2v9k7oa)  \* @return void  
[1964:](#1aeuhw3)  \*/  
[1965:](#3uei0jw)  protected function \_prepare\_digest\_auth()  
[1966:](#29jsarp)  {  
[1967:](#op2kzi)  // If whitelist is enabled it has the first chance to kick them out  
[1968:](#38oq3nb)  if ($this->config->item('rest\_ip\_whitelist\_enabled'))  
[1969:](#1nu0dv4)  {  
[1970:](#47tnwix)  $this->\_check\_whitelist\_auth();  
[1971:](#2myy6qq)  }  
[1972:](#1248gyj)   
[1973:](#3m3vzmc)  // We need to test which server authentication variable to use,  
[1974:](#21969u5)  // because the PHP ISAPI module in IIS acts different from CGI  
[1975:](#4l8tshy)  $digest\_string = $this->input->server('PHP\_AUTH\_DIGEST');  
[1976:](#30e42pr)  if ($digest\_string === NULL)  
[1977:](#1fjecxk)  {  
[1978:](#3zj1vld)  $digest\_string = $this->input->server('HTTP\_AUTHORIZATION');  
[1979:](#2eoc5t6)  }  
[1980:](#ttmg0z)   
[1981:](#3dt9yos)  $unique\_id = uniqid();  
[1982:](#1syk8wl)   
[1983:](#4cy7rke)  // The $\_SESSION['error\_prompted'] variable is used to ask the password  
[1984:](#2s3i1s7)  // again if none given or if the user enters wrong auth information  
[1985:](#178sc00)  if (empty($digest\_string))  
[1986:](#3r8funt)  {  
[1987:](#26dq4vm)  $this->\_force\_login($unique\_id);  
[1988:](#lj0f3f)  }  
[1989:](#35inxr8)   
[1990:](#1kny7z1)  // We need to retrieve authentication data from the $digest\_string variable  
[1991:](#44nlqmu)  $matches = [];  
[1992:](#2jsw0un)  preg\_match\_all('@(username|nonce|uri|nc|cnonce|qop|response)=[\'"]?([^\'",]+)@', $digest\_string, $matches);  
[1993:](#yy6b2g)  $digest = (empty($matches[1]) || empty($matches[2])) ? [] : array\_combine($matches[1], $matches[2]);  
[1994:](#3ixttq9)   
[1995:](#1y343y2)  // For digest authentication the library function should return already stored md5(username:restrealm:password) for that username see rest.php::auth\_library\_function config  
[1996:](#4i2rmlv)  $username = $this->\_check\_login($digest['username'], TRUE);  
[1997:](#2x81wto)  if (array\_key\_exists('username', $digest) === FALSE || $username === FALSE)  
[1998:](#1cdc71h)  {  
[1999:](#3wczppa)  $this->\_force\_login($unique\_id);  
[2000:](#2bi9zx3)  }  
[2001:](#qnka4w)   
[2002:](#3an7ssp)  $md5 = md5(strtoupper($this->request->method).':'.$digest['uri']);  
[2003:](#1psi30i)  $valid\_response = md5($username.':'.$digest['nonce'].':'.$digest['nc'].':'.$digest['cnonce'].':'.$digest['qop'].':'.$md5);  
[2004:](#49s5lob)   
[2005:](#2oxfvw4)  // Check if the string don't compare (case-insensitive)  
[2006:](#142q63x)  if (strcasecmp($digest['response'], $valid\_response) !== 0)  
[2007:](#3o2dorq)  {  
[2008:](#237nyzj)  // Display an error response  
[2009:](#icy97c)  $this->response([  
[2010:](#32clrv5)  $this->config->item('rest\_status\_field\_name') => FALSE,  
[2011:](#1hhw22y)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_invalid\_credentials')  
[2012:](#41hjkqr)  ], self::HTTP\_UNAUTHORIZED);  
[2013:](#2gmtuyk)  }  
[2014:](#vs456d)  }  
[2015:](#3frrnu6)   
[2016:](#1ux1y1z)  /\*\*  
[2017:](#4ewpgps)  \* Checks if the client's ip is in the 'rest\_ip\_blacklist' config and generates a 401 response  
[2018:](#2u1zqxl)  \*  
[2019:](#197a15e)  \* @access protected  
[2020:](#3t6xjt7)  \* @return void  
[2021:](#28c7u10)  \*/  
[2022:](#nhi48t)  protected function \_check\_blacklist\_auth()  
[2023:](#37h5mwm)  {  
[2024:](#1mmfx4f)  // Match an ip address in a blacklist e.g. 127.0.0.0, 0.0.0.0  
[2025:](#46m3fs8)  $pattern = sprintf('/(?:,\s\*|^)\Q%s\E(?=,\s\*|$)/m', $this->input->ip\_address());  
[2026:](#2lrdq01)   
[2027:](#10wo07u)  // Returns 1, 0 or FALSE (on error only). Therefore implicitly convert 1 to TRUE  
[2028:](#3kwbivn)  if (preg\_match($pattern, $this->config->item('rest\_ip\_blacklist')))  
[2029:](#201lt3g)  {  
[2030:](#4k19br9)  // Display an error response  
[2031:](#2z6jlz2)  $this->response([  
[2032:](#1ebtw6v)  $this->config->item('rest\_status\_field\_name') => FALSE,  
[2033:](#3ybheuo)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_ip\_denied')  
[2034:](#2dgrp2h)  ], self::HTTP\_UNAUTHORIZED);  
[2035:](#sm1zaa)  }  
[2036:](#3clphy3)  }  
[2037:](#1rqzs5w)   
[2038:](#4bqnatp)  /\*\*  
[2039:](#2qvxl1i)  \* Check if the client's ip is in the 'rest\_ip\_whitelist' config and generates a 401 response  
[2040:](#1617v9b)  \*  
[2041:](#3q0vdx4)  \* @access protected  
[2042:](#2565o4x)  \* @return void  
[2043:](#kbfycq)  \*/  
[2044:](#34b3h0j)  protected function \_check\_whitelist\_auth()  
[2045:](#1jgdr8c)  {  
[2046:](#43g19w5)  $whitelist = explode(',', $this->config->item('rest\_ip\_whitelist'));  
[2047:](#2ilbk3y)   
[2048:](#xqlubr)  array\_push($whitelist, '127.0.0.1', '0.0.0.0');  
[2049:](#3hq9czk)   
[2050:](#1wvjn7d)  foreach ($whitelist as &$ip)  
[2051:](#4gv75v6)  {  
[2052:](#2w0hg2z)  // As $ip is a reference, trim leading and trailing whitespace, then store the new value  
[2053:](#1b5rqas)  // using the reference  
[2054:](#3v5f8yl)  $ip = trim($ip);  
[2055:](#2aapj6e)  }  
[2056:](#pfzte7)   
[2057:](#39fnc20)  if (in\_array($this->input->ip\_address(), $whitelist) === FALSE)  
[2058:](#1okxm9t)  {  
[2059:](#48kl4xm)  $this->response([  
[2060:](#2npvf5f)  $this->config->item('rest\_status\_field\_name') => FALSE,  
[2061:](#12v5pd8)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_ip\_unauthorized')  
[2062:](#3mut811)  ], self::HTTP\_UNAUTHORIZED);  
[2063:](#2203i8u)  }  
[2064:](#h5dsgn)  }  
[2065:](#3151b4g)   
[2066:](#1gablc9)  /\*\*  
[2067:](#409z402)  \* Force logging in by setting the WWW-Authenticate header  
[2068:](#2ff9e7v)  \*  
[2069:](#ukjofo)  \* @access protected  
[2070:](#3ek773h)  \* @param string $nonce A server-specified data string which should be uniquely generated  
[2071:](#1tphhba)  \* each time  
[2072:](#4dp4zz3)  \* @return void  
[2073:](#2sufa6w)  \*/  
[2074:](#17zpkep)  protected function \_force\_login($nonce = '')  
[2075:](#3rzd32i)  {  
[2076:](#274ndab)  $rest\_auth = $this->config->item('rest\_auth');  
[2077:](#m9xni4)  $rest\_realm = $this->config->item('rest\_realm');  
[2078:](#369l65x)  if (strtolower($rest\_auth) === 'basic')  
[2079:](#1levgdq)  {  
[2080:](#45eiz1j)  // See http://tools.ietf.org/html/rfc2617#page-5  
[2081:](#2kjt99c)  header('WWW-Authenticate: Basic realm="'.$rest\_realm.'"');  
[2082:](#zp3jh5)  }  
[2083:](#3jor24y)  elseif (strtolower($rest\_auth) === 'digest')  
[2084:](#1yu1ccr)  {  
[2085:](#4itov0k)  // See http://tools.ietf.org/html/rfc2617#page-18  
[2086:](#2xyz58d)  header(  
[2087:](#1d49fg6)  'WWW-Authenticate: Digest realm="'.$rest\_realm  
[2088:](#3x3wy3z)  .'", qop="auth", nonce="'.$nonce  
[2089:](#2c978bs)  .'", opaque="' . md5($rest\_realm).'"');  
[2090:](#rehijl)  }  
[2091:](#3be517e)   
[2092:](#1qjfbf7)  // Display an error response  
[2093:](#4aj2u30)  $this->response([  
[2094:](#2pod4at)  $this->config->item('rest\_status\_field\_name') => FALSE,  
[2095:](#14tneim)  $this->config->item('rest\_message\_field\_name') => $this->lang->line('text\_rest\_unauthorized')  
[2096:](#3otax6f)  ], self::HTTP\_UNAUTHORIZED);  
[2097:](#23yl7e8)  }  
[2098:](#j3vhm1)   
[2099:](#333j09u)  /\*\*  
[2100:](#1i8tahn)  \* Updates the log table with the total access time  
[2101:](#428gt5g)  \*  
[2102:](#2hdr3d9)  \* @access protected  
[2103:](#wj1dl2)  \* @author Chris Kacerguis  
[2104:](#3giow8v)  \* @return bool TRUE log table updated; otherwise, FALSE  
[2105:](#1vnz6go)  \*/  
[2106:](#4fnmp4h)  protected function \_log\_access\_time()  
[2107:](#2uswzca)  {  
[2108:](#19y79k3)  $payload['rtime'] = $this->\_end\_rtime - $this->\_start\_rtime;  
[2109:](#3txus7w)   
[2110:](#29352fp)  return $this->rest->db->update(  
[2111:](#o8fcni)  $this->config->item('rest\_logs\_table'), $payload, [  
[2112:](#3882vbb)  'id' => $this->\_insert\_id  
[2113:](#1ndd5j4)  ]);  
[2114:](#47d0o6x)  }  
[2115:](#2miayeq)   
[2116:](#11nl8mj)  /\*\*  
[2117:](#3ln8rac)  \* Updates the log table with HTTP response code  
[2118:](#20sj1i5)  \*  
[2119:](#4ks6k5y)  \* @access protected  
[2120:](#2zxgudr)  \* @author Justin Chen  
[2121:](#1f2r4lk)  \* @param $http\_code int HTTP status code  
[2122:](#3z2en9d)  \* @return bool TRUE log table updated; otherwise, FALSE  
[2123:](#2e7oxh6)  \*/  
[2124:](#tcz7oz)  protected function \_log\_response\_code($http\_code)  
[2125:](#3dcmqcs)  {  
[2126:](#1shx0kl)  $payload['response\_code'] = $http\_code;  
[2127:](#4chkj8e)   
[2128:](#2rmutg7)  return $this->rest->db->update(  
[2129:](#16s53o0)  $this->config->item('rest\_logs\_table'), $payload, [  
[2130:](#3qrsmbt)  'id' => $this->\_insert\_id  
[2131:](#25x2wjm)  ]);  
[2132:](#l2d6rf)  }  
[2133:](#3520pf8)   
[2134:](#1k7azn1)  /\*\*  
[2135:](#446yiau)  \* Check to see if the API key has access to the controller and methods  
[2136:](#2jc8sin)  \*  
[2137:](#yhj2qg)  \* @access protected  
[2138:](#3ih6le9)  \* @return bool TRUE the API key has access; otherwise, FALSE  
[2139:](#1xmgvm2)  \*/  
[2140:](#4hm4e9v)  protected function \_check\_access()  
[2141:](#2wreoho)  {  
[2142:](#1bwoyph)  // If we don't want to check access, just return TRUE  
[2143:](#3vwchda)  if ($this->config->item('rest\_enable\_access') === FALSE)  
[2144:](#2b1mrl3)  {  
[2145:](#q6x1sw)  return TRUE;  
[2146:](#3a6kkgp)  }  
[2147:](#1pbuuoi)   
[2148:](#49bidcb)  // Fetch controller based on path and controller name  
[2149:](#2ogsnk4)  $controller = implode(  
[2150:](#13m2xrx)  '/', [  
[2151:](#3nlqgfq)  $this->router->directory,  
[2152:](#22r0qnj)  $this->router->class  
[2153:](#hwb0vc)  ]);  
[2154:](#31vyjj5)   
[2155:](#1h18tqy)  // Remove any double slashes for safety  
[2156:](#410wcer)  $controller = str\_replace('//', '/', $controller);  
[2157:](#2g66mmk)   
[2158:](#vbgwud)  // Query the access table and get the number of results  
[2159:](#3fb4fi6)  return $this->rest->db  
[2160:](#1ugeppz)  ->where('key', $this->rest->key)  
[2161:](#4eg28ds)  ->where('controller', $controller)  
[2162:](#2tlcill)  ->get($this->config->item('rest\_access\_table'))  
[2163:](#18qmste)  ->num\_rows() > 0;  
[2164:](#3sqabh7)  }  
[2165:](#27vklp0)   
[2166:](#n0uvwt)  /\*\*  
[2167:](#370iekm)  \* Checks allowed domains, and adds appropriate headers for HTTP access control (CORS)  
[2168:](#1m5sosf)  \*  
[2169:](#465g7g8)  \* @access protected  
[2170:](#2laqho1)  \* @return void  
[2171:](#10g0rvu)  \*/  
[2172:](#3kfoajn)  protected function \_check\_cors()  
[2173:](#1zkykrg)  {  
[2174:](#4jkm3f9)  // Convert the config items into strings  
[2175:](#2ypwdn2)  $allowed\_headers = implode(' ,', $this->config->item('allowed\_cors\_headers'));  
[2176:](#1dv6nuv)  $allowed\_methods = implode(' ,', $this->config->item('allowed\_cors\_methods'));  
[2177:](#3xuu6io)   
[2178:](#2d04gqh)  // If we want to allow any domain to access the API  
[2179:](#s5eqya)  if ($this->config->item('allow\_any\_cors\_domain') === TRUE)  
[2180:](#3c529m3)  {  
[2181:](#1racjtw)  header('Access-Control-Allow-Origin: \*');  
[2182:](#4ba02hp)  header('Access-Control-Allow-Headers: '.$allowed\_headers);  
[2183:](#2qfacpi)  header('Access-Control-Allow-Methods: '.$allowed\_methods);  
[2184:](#15kkmxb)  }  
[2185:](#3pk85l4)  else  
[2186:](#24pifsx)  {  
[2187:](#jusq0q)  // We're going to allow only certain domains access  
[2188:](#33ug8oj)  // Store the HTTP Origin header  
[2189:](#1izqiwc)  $origin = $this->input->server('HTTP\_ORIGIN');  
[2190:](#42ze1k5)  if ($origin === NULL)  
[2191:](#2i4obry)  {  
[2192:](#x9ylzr)  $origin = '';  
[2193:](#3h9m4nk)  }  
[2194:](#1wewevd)   
[2195:](#4gejxj6)  // If the origin domain is in the allowed\_cors\_origins list, then add the Access Control headers  
[2196:](#2vju7qz)  if (in\_array($origin, $this->config->item('allowed\_cors\_origins')))  
[2197:](#1ap4hys)  {  
[2198:](#3uos0ml)  header('Access-Control-Allow-Origin: '.$origin);  
[2199:](#29u2aue)  header('Access-Control-Allow-Headers: '.$allowed\_headers);  
[2200:](#ozcl27)  header('Access-Control-Allow-Methods: '.$allowed\_methods);  
[2201:](#38z03q0)  }  
[2202:](#1o4adxt)  }  
[2203:](#483xwlm)   
[2204:](#2n986tf)  // If the request HTTP method is 'OPTIONS', kill the response and send it to the client  
[2205:](#12eih18)  if ($this->input->method() === 'options')  
[2206:](#3me5zp1)  {  
[2207:](#21jg9wu)  exit;  
[2208:](#goqk4n)  }  
[2209:](#30oe2sg)  }  
[2210:](#1ftod09)   
[2211:](#3ztbvo2) }  
[2212:](#2eym5vv)

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