* Jump To … +  
    
  [<<< back to documentation](http://docs.google.com/index.html)  [base.js](http://docs.google.com/base.html)   [constraint.js](http://docs.google.com/constraint.html)   [defaults.js](http://docs.google.com/defaults.html)   [factory.js](http://docs.google.com/factory.html)   [field.js](http://docs.google.com/field.html)   [form.js](http://docs.google.com/form.html)   [main.js](http://docs.google.com/main.html)   [multiple.js](http://docs.google.com/multiple.html)   [pubsub.js](http://docs.google.com/pubsub.html)   [remote.js](http://docs.google.com/remote.html)   [ui.js](http://docs.google.com/ui.html)   [utils.js](http://docs.google.com/utils.html)   [validator.js](http://docs.google.com/validator.html)   [validator\_registry.js](http://docs.google.com/validator_registry.html)

# remote.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Utils from './utils';  
  import Base from './base';  
    
  import Parsley from './main';  
    
  $.extend(true, Parsley, {  
   asyncValidators: {  
   'default': {  
   fn: function (xhr) {
* [¶](#30j0zll)  
  By default, only status 2xx are deemed successful. Note: we use status instead of state() because responses with status 200 but invalid messages (e.g. an empty body for content type set to JSON) will result in state() === ‘rejected’.  
   return xhr.status >= 200 && xhr.status < 300;  
   },  
   url: false  
   },  
   reverse: {  
   fn: function (xhr) {
* [¶](#1fob9te)  
  If reverse option is set, a failing ajax request is considered successful  
   return xhr.status < 200 || xhr.status >= 300;  
   },  
   url: false  
   }  
   },  
    
   addAsyncValidator: function (name, fn, url, options) {  
   Parsley.asyncValidators[name] = {  
   fn: fn,  
   url: url || false,  
   options: options || {}  
   };  
    
   return this;  
   }  
    
  });  
    
  Parsley.addValidator('remote', {  
   requirementType: {  
   '': 'string',  
   'validator': 'string',  
   'reverse': 'boolean',  
   'options': 'object'  
   },  
    
   validateString: function (value, url, options, instance) {  
   var data = {};  
   var ajaxOptions;  
   var csr;  
   var validator = options.validator || (true === options.reverse ? 'reverse' : 'default');  
    
   if ('undefined' === typeof Parsley.asyncValidators[validator])  
   throw new Error('Calling an undefined async validator: `' + validator + '`');  
    
   url = Parsley.asyncValidators[validator].url || url;
* [¶](#3znysh7)  
  Fill current value  
   if (url.indexOf('{value}') > -1) {  
   url = url.replace('{value}', encodeURIComponent(value));  
   } else {  
   data[instance.element.getAttribute('name') || instance.element.getAttribute('id')] = value;  
   }
* [¶](#2et92p0)  
  Merge options passed in from the function with the ones in the attribute  
   var remoteOptions = $.extend(true, options.options || {} , Parsley.asyncValidators[validator].options);
* [¶](#tyjcwt)  
  All $.ajax(options) could be overridden or extended directly from DOM in data-parsley-remote-options  
   ajaxOptions = $.extend(true, {}, {  
   url: url,  
   data: data,  
   type: 'GET'  
   }, remoteOptions);
* [¶](#3dy6vkm)  
  Generate store key based on ajax options  
   instance.trigger('field:ajaxoptions', instance, ajaxOptions);  
    
   csr = $.param(ajaxOptions);
* [¶](#1t3h5sf)  
  Initialise querry cache  
   if ('undefined' === typeof Parsley.\_remoteCache)  
   Parsley.\_remoteCache = {};
* [¶](#4d34og8)  
  Try to retrieve stored xhr  
   var xhr = Parsley.\_remoteCache[csr] = Parsley.\_remoteCache[csr] || $.ajax(ajaxOptions);  
    
   var handleXhr = function () {  
   var result = Parsley.asyncValidators[validator].fn.call(instance, xhr, url, options);  
   if (!result) // Map falsy results to rejected promise  
   result = $.Deferred().reject();  
   return $.when(result);  
   };  
    
   return xhr.then(handleXhr, handleXhr);  
   },  
    
   priority: -1  
  });  
    
  Parsley.on('form:submit', function () {  
   Parsley.\_remoteCache = {};  
  });  
    
  Base.prototype.addAsyncValidator = function () {  
   Utils.warnOnce('Accessing the method `addAsyncValidator` through an instance is deprecated. Simply call `Parsley.addAsyncValidator(...)`');  
   return Parsley.addAsyncValidator(...arguments);  
  };