Appendix C

AjaxTCR Library Reference

Throughout this book we have been developing a library for illustrating the various ideas behind Ajax. We have incrementally added the features found in the library as we addressed each Ajax challenge. In this appendix we bring together the complete syntax found in the library with small syntax examples for guidance.

We remind readers that the AjaxTCR library is educational in focus and doesn't aim to be mission critical in its approach. The library's goals are to fully explore the issues, particularly communication-oriented ones that Ajax developers may face. While the library does include numerous useful data and utility features, we aim to provide only what is necessary for basic Ajax development and fully acknowledge that larger libraries may provide broader and richer solutions to these problems. When reviewing the library source, we encourage readers to focus on facility, clarity, and substance over personal coding preference. The overarching aim of our coding style is simplicity of algorithm over all else, so if you can understand what we are doing, you are free to do it your own way.

NOTE *The syntax reference is normative for version 1.0 of the ajaxtcr.js library. Later versions may have new methods or slight changes to syntax. Always check the book support site (ajaxref.com) for the latest syntax information.*

Coding Conventions Used

The AjaxTCR.js library uses the following basic coding practices:

- All objects, properties, and methods are encapsulated in a wrapper object `AjaxTCR`.
- Values in all CAPS are to be treated as constants (for example, `AjaxTCR.comm .DEFAULT_TRANSPORT_INDICATOR`).
- Camel case (for example, `myFavoriteMethod`) is used for properties and methods.
- Generatedvalues always use a unique ID value to avoid collisions with other scripts in page.
- If native methods or objects exist, we use those instead.
- Where possible, hard-coded values are referenced with a constant or defaults object property.

- Objects share similar names for methods: `get()`, `clear()`, `set()`, and so on.
- Methods avoid excessive parameters and employ options objects if needed.
- Private methods and values are prefixed with an underscore "_."

AjaxTCR.comm

This object provides basic features to send and abort communications requests in JavaScript, implemented generally using the `XMLHttpRequest` object, but also supporting numerous other transport mechanisms. Configuration of requests is performed by setting values of an options object described in Table C-1, though the common `setDefault()` method can be used to affect these and other constant values in a global manner.

Request Options Object Properties

When creating requests, a variety of configuration options must be set. Rather than providing numerous parameters and methods to control data transmission, an options object is utilized. Table C-2 details the settable options in the current version of the library and their defaults when nothing is specified. It should be noted that the user may set options of their own names if they desire to locally pass data values around within generated request object. Conventionally, we would suggest using a `userVars` property to perform this duty; we show that in the table, but any value is possible.

Methods	Description	Example
`abortRequest(`*`requestObj`*`)`	Aborts the XHR request of the given request object.	`AjaxTCR.comm.abortRequest(r1)`

`sendRequest(`*`URL`*
`[,`*`options`*`])`
	Primary method called to send the request. Requires a string for the URL parameter and an optional object of *options* as specified in Table C-2. If no options are specified, an asynchronous GET request is made to the URL in question, though no callback is registered to address it. When properly called, the method returns a reference to the created request object that would be used by `abortRequest()` or in some of the queue- and cache-related methods.	`var r1 = AjaxTCR.com`
		`.sendRequest("http://ajaxref`
		`.com/ch3/setrating.php",`
		`{ method: "GET",`
		`serializeForm :`
		`"ratingForm",`
		`outputTarget :`
		`"responseOutput"`
		`});`
		
		`var r2 = AjaxTCR.com.`
		`sendRequest("http://ajaxref`
		`.com/ch3/setrating.php",`
		`{ method: "POST",`
		`async : false,`
		`payload : "rating=5&comment=`
		`Love+it",`
		`outputTarget :`
		`"responseOutput"`
		`});`

setDefault (option, value)	Sets the default value for the option of interest so that it is global for all requests made.	`AjaxTCR.comm.setDefault("trans` `portIndicator",false);`

Table C-1 Public Methods for AjaxTCR.comm

Option	Description	Default	Example
`async: Boolean`	Defines if the request should be asynchronous or not. The default is true when not specified.	`true`	`async: false`
`cacheKey : string`	By default items are saved in cache using the URL of the object as a key. If another value is desired you may set it through this property, though you will be responsible for manually retrieving, as the request system will use the URL of requests to determine if something is cached or not.	`URL of request`	`cacheKey : "`
`cacheResponse: Boolean`	Boolean that indicates if the response should be saved in the response cache.	`false`	`cacheResponse: true`
`cacheTemplate: Boolean`	If a cache is returned with the response, indicates if it should be saved in the template cache or not.	`true`	`cacheTemplate: true`
`cookieName : string`	The name of the cookie expected upon response when the transport type is image. If specified, the `responseText` will be populated with the value of this cookie only. If unspecified, `responseText` will contain the entire cookie and the developer is required to parse out the response manually. Should be set if `outputTarget` is also specified with request.	`document .cookie`	`cookieName : "responsePayload"`
`enforceOrder : Boolean`	Boolean that forces every response that has this value set to be returned in the order in which it was sent; this means that requests may be held until previous requests arrive.	`false`	`enforceOrder: true`
`fallback : Boolean`	Defines if the communication mechanism should fall back to another method if the XHR fails for some reason. The fallback transport scheme is defined by `fallbackTransport`, or the global default is consulted.	`true`	`fallback: true`

TABLE C-2 Options Object Properties for Making Requests

Option	Description	Default	Example
`fallbackTransport: "iframe" | "script" | "image"`	Defines the particular communication mechanism that should be used if XHRs fail for some reason fallback. If undefined, the global default (iframe) is used unless it has been overridden.	`"iframe"`	`fallbackTransport: "image"`
`headers: Array-of-Header Objects`	An array of header objects to be sent with the request. The header object must have two properties called `name` and `value` with the appropriate values. It is set up in this manner to allow multiple values for a single name. The library will append these together with ',' . Note that setting a cookie header should be avoided, particularly if more than one value is set; `document.cookie` should be used instead.	`[]`	`headers : new Array({name: "X-Header1", value: "Value1"}, {name: "X-Header2", value: "Value2"});`

`history : object`	Controls the history mechanism on a request basis. The passed object has three properties, `saveResponse`, `id`, and `title`. The `saveResponse` property indicates that the response will be cached and when a user backs up to the page in question another request will not be issued. By default, responses will not be saved. The `id` is the value used in the hash mark (for example, `#currentState`); the `id` is required. The `title` property is used to set the title of the page so as to reflect the current state of the application.	`null`	`history : { saveResponse: true, id: "viper", title: "Technical Specifications of Colonial Viper" }`
			
			`history : { saveResponse: false, id: "add", title: "Add a to-do item" }`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example
`insertionType: "insertBefore" | "insertAfter" | "firstChild" | "lastChild" | "replace"`	Used in conjunction with `outputTarget` to define how content returned should be handled relative to the element specified by the `outputTarget` value. By default, the returned content will replace the `outputTarget` element content. Other values include: * `insertBefore` put as an element just before the specified element * `insertAfter` put as an element just after the specified element * `firstChild` put as the first child within the specified element * `lastChild` put as the last child within the specified element	`"replace"`	`outputTarget : "responseDiv", insertionType: "firstChild"`
`method: HTTP-method`	Sets the method for the request to the string *HTTP-method*. No limit to what is settable, though some XHR implementations will not support some methods and of course destinations may reject methods. If unset, a default method will be used. Note that some browsers' XHR implementations will not allow for extended HTTP methods and that alternate transfers may be even more restrictive (iframe: GET and POST; all other transports: GET only).	`"GET"`	`method: "POST"` `method: "HEAD"`
`onCreate : function`	Called right after the XHR object is created. Corresponds to `readyState == 0`. Passes the request object.	`null`	`onCreate : createFunction`
`oneway : Boolean`	Indicates if the request is one way and thus if the response should be ignored.	`false`	`oneway: true`
`onFail : function`	Callback that is called when a server error occurs. Most often this occurs when the `status != 200`. Passes the request object along with a message describing the error.	`function () {}`	`onFail : showError`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example
`onLoading : function`	Callback that is invoked when the `xhr.readyState == 3`. This occurs when the data begins to come back. Passes the request object.	`null`	`onLoading : showLoad`
`onOpen : function`	Callback that is called when the `xhr.readyState == 1`. This occurs after `xhr.open()`. Passes the request object.	`null`	`onOpen : showOpen`
`onPrefetch : function`	Callback that is invoked when you are prefetching data but not yet using it.	`function () {}`	`onPrefetch : updateCache`
`onProgress : function`	Callback invoked by default once every second. Useful for updating the user to the progress of long requests. Often used with the status object. You can override the default `progressInterval` of one second if desired.	`function () {}`	`onProgress : showProgress`
`onReceived`	Callback that corresponds to `readyState 4` but without having looked at the success or failure of the request yet, thus it will be called before `onSuccess` or `onFail`.	`null`	`onReceived : inspect`
`onRetry : function`	Callback function that is called when `retry` is enabled. Called every time a retry occurs.	`function () {}`	`onRetry : showRetry`
`onSent : function`	Callback that is called when the `xhr.readyState = 2`. This occurs right after `xhr.send()`. Passes the request object.	`null`	`onSent : showSent`
`onStatus` `: function`	Callback that is invoked for the corresponding status code. For example, the callback for `on404` is called when a response of 404 is received, while an `on500` is called when a 500 response code is received.	`undefined`	`on404 : reportBrokenLink` `on500 : stopComm`
`onSuccess : function`	Primary callback that will be called whenever the request completes successfully with a status of 200. Passes the response object as a parameter.	`function () {}`	`onSuccess : showSuccess`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example
`onTimeout : function`	Callback that is invoked when a timeout occurs. If there are retries and continual failures, this callback may be invoked numerous times.	`function () {}`	`onTimeout : showDelay`

`outputTarget : object`	When specified the request's `responseText` will be automatically inserted into the specified object using its `innerHTML` property. The object should be a reference to a DOM element or a string to be used that references an existing DOM element by its `id` attribute. The `useRaw` option can be set to `false` if a user desires to override the immediate placement of content but still use this property as a reference.	`null`	`outputTarget : "responseOutput"`
			
			or
			
			`var responseOutput = document . getElementById(" responseOutput");`
			
			`outputTarget : responseOutput;`

`password : string`	The password to be used when addressing HTTP authentication challenges. Only supported with the XHR transport.	`""`	`password : "alpha1999"`
`payload : string`	A properly encoded string (or object) to be submitted in a query string or message body depending on the HTTP method used. Various `AjaxTCR.data` methods like `encodeValue()` and `serializeForm()` may be used to quickly form a payload. The payload must be in the format in which it is going to be used.	`""`	`payload : "spacelord=Ming+of+ Mongo&evil=true"`
`preventCache : Boolean`	When set to `true`, attempts to disable caching by setting the request header to a very old date. Users may also desire to add a unique query string as well.	`false`	`preventCache: true`
`progressInterval : millisecond`	This value is used to indicate how often the request should be polled for progress updates in milliseconds. Defaults to 1 second (1000ms).	`1000`	`progressInterval : 50`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example
`requestContentType: MimeType string`	The content type on the request. If the request is a POST, it will set the request `Content-Type` header to this value. Will base form serialization on it as well.	`"application/x-www-form-urlencoded"`	`requestContentType: "application/json"`
`requestContentTransferEncoding : encodingType`	Sets the `Content-Transfer-Encoding` header on the request to the defined value.	`""`	`requestContentTransferEncoding: "base64"`
`requestSignature : string`	Indicates the header used when signing requests and will contain the contents of `signRequest` property if it is set.	`"X-Signature"`	`requestSignature : "X-Callsign"`
`retries: Boolean/number`	Indicates if a request should be retried if an error is encountered or a timeout occurs. Set to `false` or 0 to not retry failed requests. Set this value larger than 0 to indicate number of retries	`0`	`retries: 3`
`serializeForm: form`	Automatically encodes the contents of the form specified as an object, id, or name. A default encoding of `x-www-form-urlencoded` will be used unless the `requestContentType` attribute is set.	`null`	`serializeForm : ratingForm`
`showProgress : Boolean`	Setting this property to `true` indicates that the progress event will fire.	`false`	`showProgress: true`
`signRequest : "signature string"`	Used to sign a request, typically it is an MD5 hash value that will be put in the Web page when generated by a server-side program.	`null`	`signRequest:"862f011de97d4f493c3a11c589a996ee"`
`signedResponse : Boolean`	If the response is signed, the library will check the "`Content-MD5`" header in the response and compare it to an MD5 encoding of the `responseText`. If they do not match, `onFail` is called and the `responseText` is not returned.	`false`	`signedResponse: true`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example

`statusIndicator : statusObject`	Should be set to an object that contains visual display information for indicating status. At this point it supports an object with a single property progress set to an object containing type that can be either `image` or `text`. `imgSrc` is the URL of the image to use in the case `type` is set to `image`, and `text` is a string to use in the case the type is set to `text`. A `target` property is set to the DOM `id` reference of the place the status should be displayed.	`null`	`statusIndicator : {progress : {type:"image", imgSrc: "spinner.gif", target: "responseOutput"}}`
			
			`statusIndicator : {progress : {type:"text", text: "I'm loading as fast as I Can!", target: "someDiv"}}`

`template : URL | "dynamic"`	If a URL is specified, the template to apply to a response will be fetched. If the string value of "dynamic" is used, a server-side program will respond and include a template value either as a string or as URL to fetch. These values are found in the response packet in JSON format at the properties `templateText` and `templateURL`, respectively.	`null`	`template : "templates/fancypants.tpl"`
`templateRender : "client" | "server"`	String indicating if a template should be rendered on client or server; only works if the template property is set. A default value of `client` is assumed when template is set but `templateRender` is not.	`"client"`	`templateRender : "client"`
`timeout: Boolean/ number`	Indicates whether to time out or not. `False` or 0 indicates not to catch timeouts. A number greater than 0 indicates the number of milliseconds before timing out.	`false`	`timeout : 3000`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Option	Description	Default	Example
`transport : "xhr" | "iframe" | "script" | "image"`	Transport to make the request with. By default, this will be XHR though you can change it on a per request basis. The global transport can be set with `setDefault("transport", value)`, where *value* is one of the defined strings. The transport choice may change a request depending on the capabilities of the transport indicated. For example, image and script transports will not accept a POST request and will convert it into a GET if possible.	`"xhr"`	`transport : "script"`
`transportIndicator : Boolean`	Indicates if Ajax-indicating headers such as `X-Requested-By: XHR` should be included. Normally defined by value `AjaxTCR.comm.DEFAULT_TRANSPORT_INDICATOR`. Setting as an option affects only the request made; use the general the getter/setter `AjaxTCR.comm.setDefault("DEFAULT_TRANSPORT_INDICATOR", false);`.	`true`	`transportIndicator : false`
`useRaw: Boolean`	By default this is set to `true` and is consulted when `outputTarget` is set. If set to `false`, the response's payload will not be directly put into the `outputTarget`, forcing you to manually perform any decode and placement.	`true`	`useRaw : false`
`username: string`	Used to specify the username for HTTP authentication challenges issued to a request. Only usable with an XHR transport.	`""`	`username: "koenig"`

`userVars : string | number | Boolean | array | object`	Value attached to the request/response object that may contain any form of user defined data.	`undefined`	`userVars : { numDogs : 2, dogNames ["Angus", "Tucker"] }`
			
			`userVars : "I love JavaScript"`

TABLE C-2 Options Object Properties for Making Requests (*continued*)

Request Object Instance Properties

The `sendRequest()` method returns a reference to a request object that contains a number of properties that contain useful information. Often this is referred to as a response object as well, since a number of the properties are not populated until the request has become a response. Table C-3 provides the details of all these properties; example values are omitted as they are generally self-explanatory.

Property	Description
`abort`	Boolean indicating if the request has been aborted or is currently being aborted.
`endTime`	The time when the request is finished (in milliseconds).
`fail`	Contains a string indicating why a request failed ("Response Packet Compromised", etc.).
`fromCache`	Boolean indicating if the response is pulled from cache.
`httpStatus`	String containing the HTTP status code of the response. In XHR transport, corresponds to the `status` property. Will be populated with the string "200" on other transports if successful.
`httpStatusText`	String containing the HTTP status text or reason code for the response. In XHR transport, corresponds to the `XMLHttpRequest` object's `statusText` property. Will be populated with the string "OK" on other transports if successful.
`inProgress`	Boolean indicating if the request is currently in progress.
`inQueue`	Boolean indicating if the request is currently in the request queue.
`isPrefetch`	Boolean indicating that this is a prefetch request.
`rawResponseText`	When templates are used, the `responseText` will contain the rendered content (template + data); this property is used to keep the original `responseText` around.
`received`	Boolean indicating if the request has been received or not.
`requestID`	Numeric value indicating the request's ID number.
`responseText`	The raw request data returned unless a template has been specified, and then this may contain the output of the template and received data.
`responseXML`	Pointer to the `responseXML` found in the `XMLHttpRequest` object when that transport type is used. When iframe transport is used and DOM tree is seen the field may also be populated.
`retryCount`	The current count of retries that have occurred.
`startTime`	The time when the request starts.
`timespent`	Time spent during the progress of a request (transmission/receive time) used in `showProgress` mechanisms.
`totalTime`	The total time of the request as defined by `endTime - startTime` (in milliseconds).
`xhr`	Pointer to the native XHR object if that is the transport used.
`url`	The URL of the request.

Table C-3 Properties of Request/Response Objects

AjaxTCR.comm.cache

Given that the implementation of XHRs in many browsers have concerns with caches, and we cannot rely 100 percent on a browser cache as we may not be changing URLs often. The AjaxTCR library introduces a configurable JavaScript-based caching system using a simple array to address Ajax's cache woes as overviewed in Table C-4. This object is relied upon when you set `cacheResponse` in the options of a request, but it is also directly accessible by developers.

AjaxTCR.comm.cookie

Given that the AjaxTCR library supports an image-cookie fallback transport coupled with the important role cookies play in state management in Web applications, we provide a useful method to extract information out of cookies shown in Table C-5.

The order of requests and responses in Ajax applications has been shown in Chapter 6 to be of increasing importance. To address the possibility of ordering problems the AjaxTCR library supports a priority queue. The supported methods are shown in Table C-6.

AjaxTCR.comm.stats

This object provides a simple way to collect information on the quality of communications the user is experiencing. Overall statistics, plus details on failed requests, are sent to a set URL upon page unload for forensic analysis. Table C-7 provides details on this potentially illuminating feature of the AjaxTCR library.

AjaxTCR.data

Given the continuous need in an Ajax application to encode data for transmission and decode responses from such transmissions, the AjaxTCR library provides a number of helpful functions to facilitate such efforts. Table C-8 summarizes these methods.

AjaxTCR.history

Given the architectural problems that an Ajax application can experience by not modifying the URL and updating the browser's internal history mechanism, we add a number of features, as shown in Table C-9, to allow the developer to update the URL state themselves using the hash location trick. Even if developers do not plan to address this, they may find the back button guarding method at least useful to avoid accidental application bailout.

AjaxTCR.storage

This object detailed in Table C-10 provides a generic API for persisting data across page loads. In this release of the library we focus on built-in support for persistence found in Internet Explorer and Firefox with a fallback to cookies mechanism. However, it would be easy enough to add other storage providers such as Flash local shared objects (LSO), as the API presents the concept of storage in a generic way.

Method	Description	Example

`add(key,value)`	Adds any value (any valid JavaScript data item) to the internal library cache at the string specified by key. Normally not directly invoked, though provided for advanced developer use; typical action is performed via the setting of the `cacheResponse` property in a request's option object.	`AjaxTCR.comm.cache.add("rollcall",["Murray", "Jermaine","Bret"]);`
		
		`AjaxTCR.comm.cache.add("country", "New Zealand");`
		
		`AjaxTCR.comm.cache.add("/sic.php",myXhr.responseText);`

`clear()`	Clears all items from the cache.	`AjaxTCR.comm.cache(); /* cleaning up */`
`get(key)`	Retrieves the value at the passed key.	`AjaxTCR.comm.cache.get("country"); /* would return "New Zealand" */`
`getAll()`	Returns the entire cache, which is an array of cache objects each containing the following properties: `key`, `value`, `lastAccessed`, `added`, and `totalAccessed`. Useful for manual manipulation of the cache.	`var dumped = AjaxTCR.comm.cache.getAll();`
`getSize()`	Returns the length of the cache.	`alert(AjaxTCR.comm.cache.getSize());`
`remove(key)`	Removes the cached object associated with the key string passed.	`AjaxTCR.comm.cache.remove("/sic.php");`

`setOptions(options)`	Passes an object of options to override the cache defaults. The object passed may contain the following properties: `size` of the cache, as in number of entries allowed (default 100); `algorithm` (default `LRU`) to maintain the cache based upon string values "`LRU`" (Least Recently Used) [Default], "`FIFO`" (First In First Out), and "`LFU`: (Least Frequently Used); and `expires`, which is the default number of minutes to expire an item (default 60).	`AjaxTCR.comm.cache.setOptions({size: 10, algorithm: "FIFO", expires: 2});`
		
		`AjaxTCR.comm.cache.setOptions({algorithm: "LFU" });`

TABLE C-4 Methods for Handling Response Cache

Method	Description	Example
`get(name)`	Fetches the contents of the cookie specified by the passed name string.	`alert(AjaxTCR.comm.cookie.get("oreo" ));`

TABLE C-5 Cookie Handling Method(1)AjaxTCR.comm.queue

NOTE *`clear()` and `getAll()` methods for storage get all the items in the store not just values related to the current page. Use with caution.*

AjaxTCR.template

Given the need to create HTML fragments to present Ajax-provided data, we introduced a basic templating system. The template language included provides only the most basic constructs. Variables can be set and substituted, simple selections with an if construct can be used to insert markup conditional and loops utilized to perform repetitive tasks such as building out table rows. Table C-12 shows the basic syntax of the simple templating system.

NOTE *The template language supported is a subset of the Smarty template system. The goal is to allow for the same templates to be used either client or server side. It is may not be robust enough for large scale duties. Readers interested in templates are encouraged to explore one of the many client-side templating libraries emerging for more complex functionality.*

With the template defined either as a file or a string, you may populate it with data for output. Since templates are heavily used, a special caching mechanism is provided just for them. The basic methods that control these duties are shown in Table C-11.

AjaxTCR.util.DOM

We provide a basic set of DOM methods useful to more quickly select elements in Web pages and Ajax response packets containing DOM trees. The methods presented in Table C-13 provide only the most important functionality; other libraries available online may provide a much richer set of helper methods.

Method	Description	Example

`add (url,options[,priority])`	Adds the request defined by the URL and options object to the queue of requests to be made. Returns a `requestQueueID` value that can be used to remove the request from the queue. If unspecified, the priority of the request is "`normal`", which is the end of the queue. A value of "`faster`" indicates that the request should be in front of all the normal requests but at the end of any queued priority requests. A value of "`next`" puts the request at the front to be serviced next.	`var qId = AjaxTCR.comm.queue.add ("http://ajaxref.com/ch3/setrating.php",{ method: "GET", serializeForm : "ratingForm", outputTarget : "responseOutput"}, "faster" );`
		
		`AjaxTCR.comm.queue.add("http://ajaxref.com/ch1/sayhello.php",{ method: "GET", outputTarget : "responseOutput"}, "next" );`

`clear()`	Empties the entire request queue of pending requests.	`AjaxTCR.comm.queue.clear();`
`get(requestQueueID)`	Fetches the request object queued as specified by its `requestQueueID`.	`AjaxTCR.comm.queue.get(qId);`
`getAll()`	Returns an array of objects with each object having the properties URL and options that correspond to the queued requests features.	`var theLine = AjaxTCR.comm.queue.getAll()`
`getPosition(requestQueueID)`	Returns the position in the queue of the passed *`requestQueueID`* value.	`var placeInLine = AjaxTCR.comm.queue.getPosition(qId);`
`getSize()`	Returns the number of requests in the queue.	`alert("There are currently " + AjaxTCR.comm.queue.getSize() + " requests waiting to be serviced");`
`remove(requestQueueID)`	Removes the specified item from the queue. The removed item is returned if successful or false if the item is not found.	`AjaxTCR.comm.removeFromRequestQueue(qId);`

TABLE C-6 Request Queue Management Methods

Method	Description	Example
`collect(url)`	Sends the communication statistics collected to the specified URL as a JSON packet using a POST request made upon page unload. The JSON object contains `totalRequests`, `totalTimeouts`, `totalRetries`, `totalSuccesses`, `totalFails`, and `requestFails`. The `requestFails` is an array of objects where the object contains the `url`, the `status` (the HTTP status), and `message` (contains any error message).	`AjaxTCR.comm.stats.collect("collectStats.php");`
`get()`	Returns the object that is storing the statistics. The JSON object contains `totalRequests`, `totalTimeouts`, `totalRetries`, `totalSuccesses`, `totalFails`, and `requestFails`. The `requestFails` is an array of objects where the object contains the `url`, the `status` (the HTTP status), and `message` (contains any error message). Note that it will contain only values up until the time called, and any values sent by collect may include subsequent request data.	`var statusReport = AjaxTCR.comm.stats.get();`

`getRequestCount(type)`	Returns the number of requests. The `type` defaults to "`all`", which includes active requests and queued requests. The other options are "`active`" and "`queued`".	`var total = AjaxTCR.comm.stats.getRequestCount();`
		
		`var waiting = AjaxTCR.comm.stats.getRequestCount("queued");`

TABLE C-7 Communication Statistics Management Methods

Method	Description	Example

`encodeValue(`*`string`*`)`	Encodes the passed string in a properly escaped `application/x-www-form-urlencoded` manner.	`alert(AjaxTCR.data.encodeValue("Thomas O'Mallery & Sons"));`
		
		`// Thomas+O%27Mallery+%26+Sons`

`decodeValue(`*`string`*`)`	Decodes any passed value in `application/x-www-form-urlencoded` format into a standard string format.	`alert(AjaxTCR.data.decodeValue("Thomas+O%27Mallery+%26+Sons");`
		
		`// Thomas O'Mallery & Sons`

`encode64(`*`string`*`)`	Encodes the given string in base64.	`alert(AjaxTCR.data.encode64("Commodore 64s rule! "));` `// Q29tbW9kb3JlIDY0cyBydWxlIQ==`
`decode64(`*`string`*`)`	Decodes the given string from base64.	`alert(AjaxTCR.data.decode64("Q29tbW9kb3JlIDY0cyBydWxlIQ=="));` `// Commodore 64s rule!`

`encodeMD5(`*`string`*`)`	Returns the MD5 hash for the passed value.	`alert(AjaxTCR.data.encodeMD5("Victor's SD6 password"));`
		
		`// b6ff483973dd812212708f460a6494fd`

`serializeForm(`*`form,encoding[,trigger,evt]`*`)`	Inspects each element in the given form (specified by a string or an object reference) and encodes it using the encoding content-type specified. Valid content-types are "`text/xml`", "`application/json`", "`text/plain`", and "`application/x-www-form-urlencoded`" (default). The optional `trigger` parameter is the DOM element that triggers the form's submission. In the case of an image submission, it adds the `X=Xcord&Y=Ycord` values to the payload string indicating where the image was clicked. If bound to a submit button, only the submit button clicked is serialized as a value. The `evt` parameter is a JavaScript event object and is used only when the `trigger` is specified.	`var payload = AjaxTCR.data.serializeForm("myForm", "application/json");`

TABLE C-8 Useful Data Manipulation Methods for Ajax

Method	Description	Example

`serializeObject(`*`payload, object`*`[,`*`encodingString`*`])`	Loops through an object of name-value pairs and encodes each using the encoding content-type specified in the `encodingString` parameter.	`var object = {lastName : "Powell", author : true };`
		`var payload = "name=Thomas";`
		`payload = AjaxTCR.data.serializeObject( payload,object);`
		`/* name=Thomas&lastName=Powell&author=t rue */`

`encodeJSON(`*`object`*`)`	Translates the given object into a JSON string.	`var obj = {firstName : "Gaius", lastName : "Baltar" , traitor : true };`
		`var payload = AjaxTCR.data. encodeJSON(obj);`
		`/* payload = {"firstName" : "Gaius", "lastName" : "Baltar" , "traitor" : true} */`

`decodeJSON(`*`string`*`)`	Translates the given string into a JavaScript object.	`payload = '{"firstName" : "Gaius", "lastName" : "Baltar" , "traitor" : true}'`
		`var obj = AjaxTCR.data. decodeJSON(payload);`
		
		`/* {firstName : "Gaius", lastName : "Baltar" , traitor : true }; */`

`encodeAsHTML(`*`string`*`)`	Translates the tags in a string to escaped characters (`<` and `>`). The function will also translate `\n` into ` `.	`var result = AjaxTCR.data. encodeAsHTML("I <b> love \n HTML</b>!");` `/* "I <b> love   HTML </ >! " */`

`serializeXML(`*`XMLobject`*`)`	Returns any passed XML tree structure back as a string; in other words, serialized.	`Given markup like`
		`  <div id="foo">Test<em>it</em>!</div>`
		
		`var result = AjaxTCR.data. serializeXML(document. getElementById("foo"));`
		
		`/* '<div id="foo">Test<em>it</em>!</ div>' */`

TABLE C-8 Useful Data Manipulation Methods for Ajax (*continued*)

Method	Description	Example
`addToHistory(id,data,title,url,options)`	Adds an item to the browser history mechanism where *id* is the key to store the history item under, data is the data to be returned to the callback function, *title* is the title to change the page, *url* is the URL to request upon reload, and *options* is an options object used for making the request again.	`AjaxTCR.history. addToHistory("homer","","This is Boring!", "http://ajaxref.com/booorring.php" {method: "GET", payload: "name=homer" });`
`getAll()`	Returns an array of all the history objects currently stored.	`var historyCopy = AjaxTCR.history.getAll(); alert(historyCopy.length); if (historyCopy.length) { var str = "History: { id : "+ historyCopy[0].id; str+= " \n title : " + historyCopy[0].title; alert(str); }`
`getPosition()`	Returns the current numeric position in the history list.	`var pos = AjaxTCR.history.getPosition(); alert("Currently at position "+pos+" in the history list");`
`enableBackGuard([message,immediate])`	Sets the `onbeforeunload` handler so that users don't accidentally leave the application. If set, the scheme will not be invoked until the user has performed their first action in case they really did want to immediately leave. Passing the optional Boolean `immediate` set to true turns the protection on before any requests are made. An optional string `message` can also be passed; otherwise, the browser will confirm solely with its standard prompt.	`AjaxTCR.history.enableBackGuard("",true); /* enable immediately */ AjaxTCR.history.enableBackGuard("Really? Leave now? ");`
`init(callback)`	Method that must be called from client in order to initializes the history mechanism. Also checks current hash, so ideal to call on page load for bookmark purposes. The callback is called when the initializing page is backed up to. The callback is also run anytime a user manually adds items to the history with the `addToHistory()` method rather than allowing the XHR to do this.	`AjaxTCR.history.init(putBackTogether);`

TABLE C-9 Methods of the AjaxTCR History Object

Method	Description	Example
`add(key,value, persistenceObject [,store])`	Stores the value specified as a string at the key specified in the appropriate data store. The `persistenceObject` is returned from the `init()` method. In the case of Internet Explorer a store parameter string may be passed as well to define what store the data is associated with.	`AjaxTCR.storage.add("fortknox","lots of gold",persistObj)`
`get(key,persistenceObject [,store])`	Retrieves data from the storage system related to the passed key and the passed `persistenceObject`. In the case of Internet Explorer you may also pass in a store value.	`var treasure = AjaxTCR.storage.get("fortknox", persistObj);` `alert(treasure);  /* shows "lots of gold" */`
`getAll(persistenceObject [,store])`	Retrieves all data from the storage system referenced by the passed `persistenceObject`. In the case of IE it would fetch only the values from the passed store value or its default value of "AjaxTCRStore" if not passed.	`var allTreasure = AjaxTCR.storage. getAll(persistObj);`
`init()`	Initializes the data store for holding persisted data. Returns a handle to the persistence object.	`var persistObj = AjaxTCR.storage.init();`
`clear(persistenceObject [, store])`	Clears all the items out of the storage system related to the passed `persistenceObject` and store value.	`AjaxTCR.storage.clear(persistObj)`
`remove(key,persistance Object[,store])`	Removes the data from the storage system related to `peristanceObject` related to the passed key. In the case of Internet Explorer you may also pass a `store` value.	`AjaxTCR.storage.remove("fortknox",persistObj);` `/* no more gold */`

TABLE C-10 Abstract Methods for Data Persistence

Method	Description
`cache(URL, [template-string])`	Fetches the template at the specified URL or specifies the template-string as the cache object for the indicated URL.

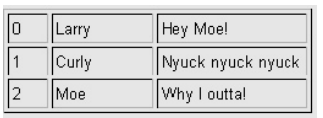
`cacheBundle(URL)`	Fetches a template from the specified URL that contains a bundle of templates to be parsed into individual pieces. Templates are separated by HTML comments like so
	
	`<!-- Template-Begin URL="URL " -->`
	`Template contents`
	`<-- Template-End -->`

`clearCache([URL])`	Removes the specified URL from the template cache or, without any parameters, all templates in the template cache.
`translateFile(templatefilename, data)`	Function takes a template as URL to the template in question and applies any passed data in the form of a JSON string to the template values converting the template to its final rendered output.
`translateString(templatestring, data)`	Function takes a template as a string of the template and applies any passed data in the form of a JSON string to the template values converting the template to its final rendered output.

Table C-11 Template Handling Methods

Construct	Description	Example Template
{$varname}	Replaces the token with the property in a JSON packet of the same value as *varname*.	{$character} says <q>{$phrase}</q>
{foreach item=*iteratingvar* from=*varname* [key=*keyval*] } *Markup-loop* [{foreachelse} *Markup-no-loop* {/foreach}	Loops through the data from *varname*, placing each value in the *iteratingvar* and outputting it against the contents of *Markup-loop*. If *keyval* is specified, it can be looked for understanding position of the current item being iterated in *varname*. Useful for "zebra striping" a table. If no data is found in varname, the *contents* of *Markup-no-loop* will be used instead.	<table border="1" cellpadding="3" cellspacing="3" width="400px"> {foreach item=stooge key=stoogenumber from=$stooges} <tr> <td>{$stoogenumber}</td> <td>{$stooge.name}</td> <td>{$stooge.line}</td> </tr> {/foreach} </table>
{if *expression*} *Markup-true* [{else}] *Markup-false* {/if}	If the value of the expression is true, output contents *Markup-true*, which may include more template constructs. If the `else` is specified and the value is value, output *Markup-false* instead.	{if $spy = "007"} Bond,..James Bond {else} Not a movie spy {/if}
{include file="URL" }	Includes a template file from the URL specified. Used simply as a stub since this will commonly be found in a server side template.	{include file="footer.tpl"}

Table C-12 Summary of Basic AjaxTCR Template Constructs

Example JSON Data	Rendering
`{"character": "Captain Kirk" , "phrase" : "To boldly go where no man has gone before!"}`	Captain Kirk says "*To boldly go where no man as gone before!*"
`{"stooges":[ {"name":"Larry", "line": "Hey Moe!"}, {"name":"Curly", "line": "Nyuck nyuck nyuck"}, {"name":"Moe", "line": "Why I outta!"} ] }`	
`{"spy" : "007"}`	Bond,...James Bond

N/A	Renders only server side but might look something like:
	
	`<!-- contents of footer .tpl -->`
	`<hr />`
	`<span class="legal">&copy; 2008 PINT, Inc.</span>`

Method	Shorthand	Description
`getElementById(id [,startNode,deepSearch])`	None	Returns a single DOM element that matches the `id` passed as a string; otherwise a null value is returned. A `startNode` can be passed to indicate where the search begins from; otherwise the document is assumed. The Boolean parameter `deepSearch` can be set to `true` to perform a brute force search of `id` attribute values that may be useful when addressing XML trees as commonly found in Ajax response packets.
`getElementsById(id [,startNode,deepSearch])`	`$id()`	Returns a single DOM element or list of DOM elements that match the `id`(s) passed as strings. A `startNode` can be passed to indicate where the search begins from; otherwise the document is assumed. The Boolean parameter `deepSearch` can be set to true to perform a brute force search of `id` attribute values that may be useful when addressing XML trees as commonly found in Ajax response packets.
`getElementsByClassName(className [,startNode])`	`$class()`	Returns a list of all the DOM elements with the specified class name. More qualified searches, such as for the stem of a class name, should use the `getElementsBySelector()` method instead.
`getElementsBySelector(selector [,startNode])`	`$selector()`	Finds all the DOM elements matching the `selector` string passed starting from the `startNode` or the document root if not specified. The `selector` string should be a string that is a well-formed CSS2 selector rule.
`insertAfter(parentNode,nodeToInsert,insertPoint)`	None	Inserts the DOM node specified by *nodeToInsert* after the node specified by *insertPoint*. The *parentNode* this operates on must be passed for reference.

TABLE C-13 Useful DOM Methods

AjaxTCR.util.event

The simple object summarized in Table C-14 is generally a stub for a more full fledged event management system to be added by the reader or to be included in a future update to the library. The only method currently included is for setting load events for the window, given how that is generally useful for binding DOM elements to event handling functions.

Method	Shorthand	Description
`addWindowLoadEvent(code)`	`$onload()`	Adds an event handler for the object specified by the string.

Table C-14 Event Management Methods

AjaxTCR.util.misc

This like to change object holds miscellaneous utilities that do not fit anywhere else and might be used in a more global sense. Table C-15 shows the single method found in this object, but it is quite likely more have found their way into the library by the time you read this so check the support site to be sure.

Method	Description
`generateUID([prefix])`	Generates a unique `id` value (UID) using current time in milliseconds with a random number appended. The prefix value is an optional string to indicate a prefix for the UID value returned. If the parameter is not set, the string "AjaxTCR" is used to further protect against any collisions if other UID generators are in play, as well as to make the UID be valid for use as a DOM value that may not start with a number. A passed prefix value of –1 will keep the prefix from being applied.

Table C-15 Miscellaneous Utility Methods