Wakanda proposes a server-side XMLHttpRequest API that allows the Wakanda server to send requests to other HTTP servers. The Wakanda Server-side implementation of XMLHttpRequest is partially compliant with the one available on client-side. Its API follows the W3C XMLHttpRequest API specification and provides an additional support for proxy connections.

The Server-side XMLHttpRequest API can be broken in three parts:

- the XMLHttpRequest( ) constructor, used to create XMLHttpRequest instances,
- the instance properties and methods to manage requests: readyState, onreadystatechange, open( ), send( ), setClientCertificate( ), setRequestHeader( )
- the instance properties and methods to manage responses: response, responseText, responseType, status, statusText, getAllResponseHeaders( ), getResponseHeader( )
XMLHttpRequest Constructor

XMLHttpRequest( )

```javascript
void XMLHttpRequest([Object proxy])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>Object</td>
<td>Object containing a host and a port attributes</td>
</tr>
</tbody>
</table>

Description

The XMLHttpRequest( ) method is the constructor of the class objects of the XMLHttpRequest type. It should be used with the `new` operator to create XMLHttpRequest instances on the server. Once created, an instance can be managed using methods and properties regarding the request itself as well as the response (see XMLHttpRequest Instances).

If the Wakanda Server needs to perform the request through a proxy server, by default the method automatically uses the system proxy settings.

**Note:** Proxy exceptions are supported through simple string comparisons.

However, if you want to override your system settings, you can pass an object containing two attributes in the `proxy` parameter:

- host (string): address of the proxy server
- port (number): TCP port number of the proxy server

For example, this object is a valid `proxy` parameter:

```javascript
{host: "http://proxy.myserver.com", port: 80}
```

If you do not want to use your proxy for the request, pass an empty object in `proxy`.

**Note:** XMLHttpRequest() supports HTTPS connections but does not validate certificates.

Keep-alive

Wakanda Server implements an automatic keep-alive mode when issuing a XMLHttpRequest(). When several successive XHR requests are sent to the same host, Wakanda Server will reuse the same TCP connection. This feature provides better performance when using an authenticated connection or when sending a large number of XHR requests.

Example

In the following example, we send GET requests to a Wakanda server or to an HTTP server and format the responses, whatever their type (HTML or JSON). We can then see the results in the JavaScript Editor.

```javascript
var xhr, headers, result, resultObj, URLText, URLJson;
URLJson = "http://127.0.0.1:8081/rest/$catalog"; // REST query to a Wakanda Server
URLText = "http://communityjs.org/"; // connect to an HTTP server
var headersObj = {};
xhr = new XMLHttpRequest(); // instanciate the xhr object
// you could pass a proxy parameter if you do not want to use your default proxy settings
xhr.onreadystatechange = function() { // event handler
    var state = this.readyState;
    if (state !== 4) { // while the status event is not Done we continue
        return;
    }
    var headers = this.getAllResponseHeaders(); //get the headers of the response
    var result = this.responseText; //get the contents of the response
    var headersArray = headers.split('\n'); // split and format the headers string in an array
    headersArray.forEach(function(header, index, headersArray) {
        var name, indexSeparator, value;
        if (header.indexOf('HTTP/1.1') === 0) { // this is not a header but a status
            return; // filter it
        }
        indexSeparator = header.indexOf(':');
        name = header.substr(0, indexSeparator);
        if (name === "") {
            return;
        }
        value = header.substr(indexSeparator + 1).trim(); // clean up the header attribute
        headersObj[name] = value; // fills an object with the headers
    });
    if (headersObj['Content-Type'] && headersObj['Content-Type'].indexOf('json') !== -1) {
        // JSON response, parse it as objects
    }
}
```
resultObj = JSON.parse(result);
} else { // not JSON, return text
    resultTxt = result;
}
}

xhr.open('GET', URLText); // to connect to a Web site
// or xhr.open('GET', URLJson) to send a REST query to a Wakanda Server
xhr.send(); // send the request
statusLine = xhr.status + ' ' + xhr.statusText; // get the status

// we build the following object to display the responses in the JavaScript Editor
{
    statusLine: statusLine,
    headers: headersObj,
    result: resultObj || resultTxt
}

The results can be displayed in the Results area of the JavaScript Editor.
Here is the result of a simple query to a Web server:

<table>
<thead>
<tr>
<th>statusLine: &quot;200 OK&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers:</td>
</tr>
<tr>
<td>Age: &quot;523&quot;</td>
</tr>
<tr>
<td>Connection: &quot;Keep-Alive&quot;</td>
</tr>
<tr>
<td>Content-Length: &quot;11800&quot;</td>
</tr>
<tr>
<td>Content-Type: &quot;text/html; charset=utf-8&quot;</td>
</tr>
<tr>
<td>Date: &quot;Fri, 06 Jan 2012 15:05:26 GMT&quot;</td>
</tr>
<tr>
<td>Proxy-Connection: &quot;Keep-Alive&quot;</td>
</tr>
<tr>
<td>Via: &quot;1.1 PROX&quot;</td>
</tr>
<tr>
<td>X-Powered-By: &quot;Express&quot;</td>
</tr>
<tr>
<td>result: &quot;CommunityJS.org&quot;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>• User Groups</td>
</tr>
<tr>
<td>• BeerJS</td>
</tr>
<tr>
<td>• About</td>
</tr>
</tbody>
</table>

CommunityJS.org
JavaScript User Groups & Conferences

Here is the result of a REST query to a Wakanda Server:
<table>
<thead>
<tr>
<th>name</th>
<th>uri</th>
<th>dataURI</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td><a href="http://127.0.0.1:8081/rest/Scatalog/City">http://127.0.0.1:8081/rest/Scatalog/City</a></td>
<td><a href="http://127.0.0.1:8081/rest/City">http://127.0.0.1:8081/rest/City</a></td>
</tr>
<tr>
<td>Comp</td>
<td><a href="http://127.0.0.1:8081/rest/Scatalog/Comp">http://127.0.0.1:8081/rest/Scatalog/Comp</a></td>
<td><a href="http://127.0.0.1:8081/rest/Comp">http://127.0.0.1:8081/rest/Comp</a></td>
</tr>
<tr>
<td>Person</td>
<td><a href="http://127.0.0.1:8081/rest/Scatalog/Person">http://127.0.0.1:8081/rest/Scatalog/Person</a></td>
<td><a href="http://127.0.0.1:8081/rest/Person">http://127.0.0.1:8081/rest/Person</a></td>
</tr>
</tbody>
</table>
**XMLHttpRequest Instances**

**status**

**Description**

The `status` property returns the HTTP status code of the `XMLHttpRequest`. `status` returns 0 if an error occurred or if the request `readyState` value is 0 or 1.

**statusText**

**Description**

The `statusText` property returns the HTTP status text of the `XMLHttpRequest`. `statusText` returns an empty string if an error occurred or if the request `readyState` value is 0 or 1.

**responseText**

**Description**

The `responseText` property contains the text response entity body. The response entity body is the fragment of the entity body of the response received so far (`readyState` value 3) or the complete entity body of the response (`readyState` value 4). The `responseText` property returns an empty string if the `readyState` value is not 3 or 4, or if the response entity body is null.

**readyState**

**Description**

The `readyState` property returns the current state of the `XMLHttpRequest`. The returned value can be one of the following:

<table>
<thead>
<tr>
<th>State value</th>
<th>State description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (UNSET)</td>
<td>The <code>XMLHttpRequest</code> object has been constructed.</td>
</tr>
<tr>
<td>1 (OPENED)</td>
<td>The <code>open()</code> method has been successfully invoked. During this state, request headers can be set using <code>setRequestHeader()</code> and the request can be made using the <code>send()</code> method.</td>
</tr>
<tr>
<td>2 (HEADERS_RECEIVED)</td>
<td>All HTTP headers have been received. Several response members of the object are now available.</td>
</tr>
<tr>
<td>3 (LOADING)</td>
<td>The response entity body is being received.</td>
</tr>
<tr>
<td>4 (DONE)</td>
<td>The data transfer has been successfully completed or something went wrong during the transfer (e.g. infinite redirects).</td>
</tr>
</tbody>
</table>

Each time the `readyState` property changes, the event handler function set by `onreadystatechange` is called.

**responseType**

**Description**

The `responseType` property allows you to set the type of the response. Two values are available:

- "text", which is the default value
- "blob", to set the response object as a BLOB.

Note that if the response entity body is of the BLOB type, you should handle it using the `response` property instead of `responseText`.

You can set `responseType` at any moment, whatever the xhr state.

**Example**

```javascript
var proxy = { host: 'myproxy.public.4d.com', port: 80 };

xhr = new XMLHttpRequest(proxy);
xhr.open('GET', 'http://www.perdu.com');
xhr.send();

//response type defaults to "text"
var typel=xhr.responseText;
var resp1=xhr.response;

//we can change responseType at will
```
xhr.responseType="blob";
var type2=xhr.responseType;
var resp2=xhr.response;

var type3=xhr.responseType;
   //we can't affect anything to response
xhr.response="this is the response";
var resp3=xhr.response;

Description

The response property contains the response entity body. The response entity body is the fragment of the entity body of the response received so far (readyState value 3) or the complete entity body of the response (readyState value 4).

You need to use this property if the responseType is other than text, otherwise you can use the responseText property. The response property returns an empty string if the readyState value is not 3 or 4, or if the response entity body is null.

Example

The following loadRemoteImage(url[, proxy]) function is an alternative to loadImage() to load an image from the Web using an HTTP URL.

* @method loadRemoteImage
* @param {string} url A full HTTP url
* @param {Object} [proxy] Proxy object expects 2 properties: "host" and "port"
* @result Image|null
**/

function loadRemoteImage(url, proxy) {
  'use strict';

  var
      xhr,
      format,
      tmpFile,
      image;

  // download remote image
  xhr = new XMLHttpRequest(proxy);
  xhr.open('GET', url, false); // false explicitly say request is synchronous
  xhr.setRequestHeader('Accept', 'image/*');
  xhr.responseType = 'blob';
  xhr.send();
  // handle redirections
  if ([301, 302, 303, 307, 308].indexOf(xhr.status) > -1) {
    url = xhr.getResponseHeader('Location');
    return loadRemoteImage(url, proxy);
  }
  if ((xhr.status !== 200) || !xhr.response || !xhr.response.size) {
    return null;
  }

  // check and transform mime type name into extension for supported format
  format = xhr.getResponseHeader('Content-Type').split('/'); // ex: ['image', 'png']
  if (format[0] !== 'image') {
    return null;
  }
  format = format[1];
  // remove potential "x-" prefix
  if (format[0] === 'x') {
    format = format.substr(2);
  }

  // save downloaded image to temporary file
  tmpFile = new File(ds.getTempFolder(), 'webImg' + Date.now() + '.' + format);
  xhr.response.copyTo(tmpFile);

  // create the image instance
  image = loadImage(tmpFile);

  // remove temporary file
  tmpFile.remove();
onreadystatechange

Description

The `onreadystatechange` property defines the event listener function that will handle the various states of the `XMLHttprequest`. The `onreadystatechange` function will be called each time the `readyState` property value is updated and receives contextual information about the event. You can handle the request in response to the current `readyState`.

Example

See the example for the `XMLHttprequest()` constructor function.

getAllResponseHeaders( )

String `getAllResponseHeaders()`

Returns String Header of the response with all its fields in plain text

Description

The `getAllResponseHeaders()` method returns all HTTP headers from the response of the `XMLHttprequest`. HTTP headers are returned as a single string with each header line separated by a CR/LF pair and with each header name and header value separated by a ";" pair.

Example

See example for the `XMLHttprequest()` constructor method.

generateResponseHeader( )

String | Null `generateResponseHeader(String header)`

Parameter | Type | Description
--- | --- | ---
header | String | Header field name
Returns | String, Null | Value of the header

Description

The `generateResponseHeader()` method returns the value of a specific `header` field in the response of the `XMLHttpRequest`. Pass in `header` the name of the header field name that you want to get the value. The method returns `Null` if an error occurred or if no such field name was found in the response.

Example

You want to know the value of the 'Content-Type' field to process the result accordingly:

```javascript
var xhr = new XMLHttpRequest();
xhr.open("GET", "http://www.myserver.com");
xhr.send();
xhr.onreadystatechange = function() {
  if(this.readyState== 4) && (this.status == 200) {
    if( this.getResponseHeader("Content-Type") == "application/json") {
      resultObj = JSON.parse(this.responseText);
    }
  } // process resultObj
} ;
```

open( )

void `open(String method , String url [, Boolean async])`

Parameter | Type | Description
--- | --- | ---
method | String | The method of the request (GET, POST, PUT, HEAD)
url | String | URL of the request
async | Boolean | False or omitted = Synchronous execution
 | | True = Asynchronous execution (not implemented)

Description
The `open()` method declares the HTTP method and the URL of the `XMLHttpRequest`.
Pass in `method` the HTTP method to use. The following standard HTTP methods are currently supported: GET, POST, PUT and HEAD.
Pass in `url` a valid URL where the request will be addressed.

**Note:** The `async` parameter is available for compatibility reasons but has no effect since only synchronous execution is currently supported in Wakanda server-side XHR.

**Example**

We just want the server to connect using HTTPS protocol and display the result in the code editor:

```javascript
xhr = new XMLHttpRequest();
xhr.open('GET','https://docs.google.com/'); // connect using the https protocol
xhr.send();
xhr.responseText;// displays the response
```

### send()

```javascript
void send([String | File data])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>String, File</td>
<td>The content of the request body for POST and PUT requests</td>
</tr>
</tbody>
</table>

**Description**

The `send()` method sends the request defined in the `XMLHttpRequest`.
The optional `data` parameter allows you to provide the request entity body. You can pass either a `string` or a `File` object. If you pass a `File`, it must already exist on disk.

This parameter is ignored if request method is GET or HEAD. Developers are encouraged to ensure that they have specified the Content-Type header using `setRequestHeader()` before invoking `send()` with a non-null `data` argument.

**Example**

The following function allows you to upload an image using a XHR:

```javascript
function uploadFile(file, fileName, containerName, authObject){
    var response = null;
    var client = new XMLHttpRequest();
    client.open('PUT', authObject.storageUrl + "" + containerName + "" + fileName);
    client.setRequestHeader('X-Auth-Token', authObj.authToken);
    client.setRequestHeader('Content-Type', "image/jpeg");
```
client.send(file);
var state = client.readyState;
if(state == 4){
    response = client.status;
}
return response;
};

**setClientCertificate( )**

```java
void setClientCertificate( String keyPath, String certificatePath )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyPath</td>
<td>String</td>
<td>Path to the PEM format private key</td>
</tr>
<tr>
<td>certificatePath</td>
<td>String</td>
<td>Path to the local PEM format certificate</td>
</tr>
</tbody>
</table>

**Description**

The `setClientCertificate( )` method allows the request to be authenticated on the remote server with a client certificate, when necessary.

Pass in `keyPath` the path to the local key in PEM format.

Pass in `certificatePath` the path to the local certificate, also in PEM format.

**setRequestHeader( )**

```java
void setRequestHeader( String header, String value )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>String</td>
<td>The header field name. Ex: &quot;Accept&quot;</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The header field value. Ex: &quot;application/json&quot;</td>
</tr>
</tbody>
</table>

**Description**

The `setRequestHeader( )` allows you to set the `value` of a specific `header` field of the `XMLHttpRequest`.

This method is useful to define custom headers or to set standard header values.

**Example**

The following script:

```javascript
var client = new XMLHttpRequest();
client.open('GET', 'demo.cgi');
client.setRequestHeader('X-Test', 'one');
client.setRequestHeader('X-Test', 'two');
client.send();
```

would result in the following header being sent:

```
X-Test: one, two
```

**Example**

In this example, you set the User-Agent:

```javascript
client.setRequestHeader('User-Agent','wakandaDB sample application');
```