XMLHttpRequest

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,
- the instance properties and methods to manage responses.

XMLHttpRequest Constructor

XMLHttpRequest( )

void **XMLHttpRequest**([Object *proxy*])

Parameter	Type	Description
proxy	Object	Object containing a host and a port attributes

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 (see **XMLHttpRequest Instances (Requests)**) as well as the response (see **XMLHttpRequest Instances (Responses)**).

If the Wakanda Server needs to perform the request through a proxy server, pass in the *proxy* parameter an object containing two attributes:

- 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:

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

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

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 Wakanda code editor.

```
var xhr, headers, result, resultObj, URLText, URLJson;
 var proxy = { // define a proxy only if necessary
     host: 'proxy.myserver.com', // use any valid proxy address
     port: 80
 }

 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(proxy); // instanciate the xhr object
    // the proxy parameter may not be necessary

 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
     headersArray.forEach(function(header, index, headersArray) {
         var name, indexSeparator, value;

        if (name.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 attribut
        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 code editor
({
    statusLine: statusLine,
    headers: headersObj,
    result: resultObj || resultTxt
});
```

The results can be displayed in the Results area of the Wakanda Studio code editor.

Here is the result of a simple query to an Web server:

Here is the result of a REST query to an Wakanda server:

statusLine: "200 OK"

headers:

Accept-Ranges: "none"

Connection: "keep-alive"

Content-Length: "350"

Content-Type: "application/json"

Date: "Fri, 06 Jan 2012 15:58:00 GMT"

Server: "Wakanda/1.0.0"

result:

dataClasses:

name: "City"	**uri:** "http://127.0.0.1:8081/rest/$catalog/City"	**dataURI:** "http://127.0.0.1:8081/rest/City"
name: "Comp"	**uri:** "http://127.0.0.1:8081/rest/$catalog/Comp"	**dataURI:** "http://127.0.0.1:8081/rest/Comp"
name: "Person"	**uri:** "http://127.0.0.1:8081/rest/$catalog/Person"	**dataURI:** "http://127.0.0.1:8081/rest/Person"

XMLHttpRequest Instances (Requests)

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.

readyState

Description

The **readyState** property returns the current state of the *XMLHttpRequest.*

The returned value can be one of the following:

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

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

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 adressed.

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.*

send( )

void **send**([String *data*])

Parameter	Type	Description
data	String	The content of the request body for POST and PUT requests

Description

The **send( )** method initiates the request defined in the *XMLHttpRequest*.

The optional *data* parameter allows you to provide the request entity body. 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.

setRequestHeader( )

void **setRequestHeader**(String *header*, String *value*)

Parameter	Type	Description
header	String	The header field name. Ex: "Accept"
value	String	The header field value. Ex: "application/json"

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:

```
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
...
```

XMLHttpRequest Instances (Responses)

status

Description

The **status** property returns the HTTP status code of the *XMLHttpRequest*.

status returns 0 if an error occured 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 occured or if the request **readyState** value is 0 or 1.

responseText

Description

The **responseText** property returns the text response entity body. The response entity body is the fragment of the entity body of the response received so far (value 3) or the complete entity body of the response (value 4)

The **responseText** property returns an empty string if the value is not 3 or 4, or if the response entity body is null.

getAllResponseHeaders( )

String **getAllResponseHeaders**( )

Returns	String	Header of the reponse 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 constructor method.

getResponseHeader( )

String | Null **getResponseHeader**(String *header*)

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

Description

The **getResponseHeader( )** 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:

```
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
 };
```