WebSocket Server

Wakanda Server provides a WebSocket Server API, allowing you to handle client WebSocket connections on the server. WebSockets enable Web applications (clients) to use the WebSocket protocol for two-way communication with a remote host (server).

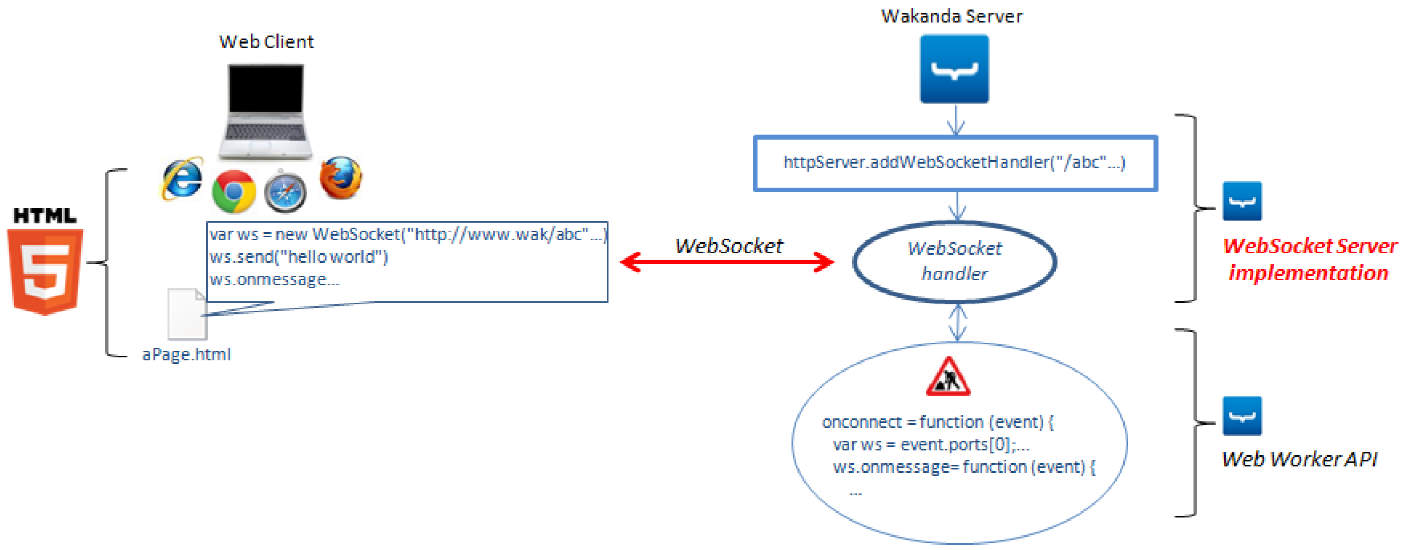
For more information on the WebSocket protocol, please refer to the WebSocket W3C specification.

Note: *WebSockets are part of HTML5. Please keep in mind that the WebSocket specification is still under discussion and should neither be considered as frozen nor as finished.*

WebSockets protocol implementation relies on different parts, implying both client and server sides:

- Client-side, WebSockets are supported through HTML5 implementation.
- On Wakanda Server, WebSocket handlers are registered using a specific API and WebSocket server instances rely on the **Web Workers** API.

To understand the Wakanda Server-side WebSocket support, it is important to identify the different parts and how they interact:


Web Client
HTML
var ws = new WebSocket("http://www.wak/abc"...)
ws.send("hello world")
ws.onmessage...
aPage.html
WebSocket
Wakanda Server
httpServer.addWebSocketHandler("/abc"...)
WebSocket handler
onconnect = function (event) {
var ws = event.ports[0];...
ws.onmessage= function (event) {
...
WebSocket Server implementation
Web Worker API

Managing server-side WebSockets

Understanding Server-side Client WebSocket Representation

On Wakanda Server, WebSockets registered by **addWebSocketHandler( )** are handled through **Web Workers**. When the server communicates with a client, the server has access to a WebSocket "proxy" object describing the client WebSocket. This proxy object is actually a Web worker, that you can handle through the regular **Web Workers** API.

To handle a websocket, you can use:

- either a **shared** worker,
- or a **dedicated** worker.

Usually, shared workers are more appropriate for handling WebSockets. However, you can select the type of worker depending on your needs while registering the WebSocket using **addWebSocketHandler( )**. APIs for shared and dedicated workers are slightly different.

Using a Shared Worker for the WebSocket

When you have defined a server-side WebSocket based on a shared worker, you can use the regular **SharedWorker Instances** APIs in the server script.

- *SharedWorker* properties are documented in the **SharedWorker Instances** API reference,
- Communication tools available through the **event.ports[0]** objects (**onconnect** and **ports**) are detailed in the **Worker Instances** API reference.

Note that two specific APIs have been added to the **Worker Instances** class to handle WebSockets:

- **binaryType** allows you to define the type of data exchanged through the WebSocket.
- **onclose** function is called each time the WebSocket is closed.

The following code structure can be used:

```
onconnect = function (event) { //Called each time a new client is connected
    var webSocket = event.ports[0]; //Access to the WebSocket client object.
            // Undefined if shared worker is called from SSJS server.
    webSocket.binaryType = 'string'; // Defines the exchanged data type
        // this worker property is only available in the context of a WebSocket
    webSocket.postMessage("helloWorld");
    webSocket.onmessage = function(message) { //Called each time a client sends a message
        var data = JSON.parse(message.data); //Application protocol
        webSocket.postMessage('Message received');
    };
    webSocket.onclose = function() { // when the socket is closed
    };
};
```

Server-side communication with the Shared Worker

If you use a shared worker to handle the WebSocket, a server-side script can communicate with it like with any other web worker:

```
var worker = new SharedWorker("chat-server.js", "chat");
worker.port.postMessage('');
    // The message will be received in the onmessage callback of the web worker
```

Using a Dedicated Worker for the WebSocket

When you have defined a server-side WebSocket based on a dedicated worker, you can use the regular **Worker Instances** APIs in the server script. In accordance with dedicated workers mechanism, the server-side worker is automatically executed each time a message is sent by the client.

Here is the list of available APIs for a server-side WebSocket based on a dedicated worker:

Worker APIs	Only for WebSocket objects
onmessage	

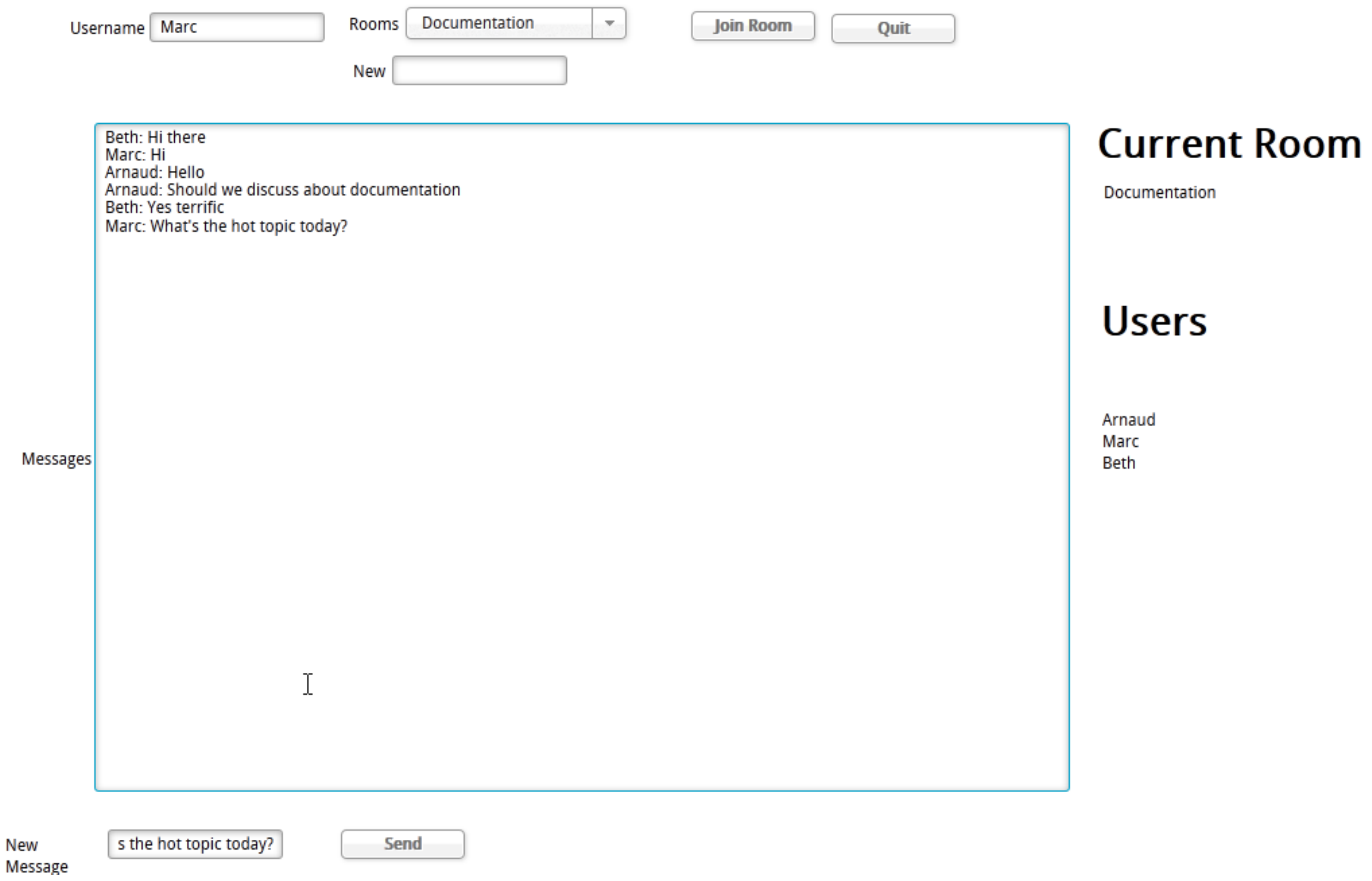
postMessage( )	
binaryType	X
onclose	X

You can use the following code structure:

```
onmessage = function(message) { //Called each time a client sends a message
    var data = JSON.parse(message.data) //Get the message
 };
postmessage("helloWorld"); //Send a message
onclose = function(); // when the socket is closed
```

Downloadable Example

A typical example of server-side WebSocket implementation is a "chat" server providing the ability for several client users to chat together in real time in rooms located on the server:



Username: Marc
Rooms: Documentation

Beth: Hi there
Marc: Hi
Arnaud: Hello
Arnaud: Should we discuss about documentation
Beth: Yes terrific
Marc: What's the hot topic today?

Current Room: Documentation

Users:
Arnaud
Marc
Beth

New Message: s the hot topic today?

Download the chat example application

This application contains client-side WebSocket HTML5 as well as server-side WebSocket implementation using shared workers. The JavaScript code is commented to explain the main techniques in use.

WebSocket Handlers

On Wakanda Server, WebSockets are managed through a specific HTTP handler. You need to install this handler and register the WebSocket using the **addWebSocketHandler( )** method from the **HTTP Server** class. WebSocket handlers can be removed using the **removeWebSocketHandler( )** method.

addWebSocketHandler( )

void **addWebSocketHandler**(String *pattern*, String *filePath*, String *socketID*, Boolean *shared*)

Parameter	Type	Description
pattern	String	Pattern or path to handle
filePath	String	Path to the JavaScript file in which the handler function is defined
socketID	String	Local name of the WebSocket
shared	Boolean	Use a shared worker (true) or a dedicated worker (false)

Description

The **addWebSocketHandler( )** method installs a WebSocket handler script on the server. Once installed, this script will be called to handle any incoming request matching the predefined *pattern*.

This method should usually be called in the **Bootstrap** file of the application.

- In the *pattern* parameter, pass a string describing the path or the pattern of the client WebSocket requests that you want to intercept. This pattern corresponds to the *url* property of the WebSocket instance on the client side. The pattern can be defined through a Regex (Regular expression). For more information, please refer to the **addHttpRequestHandler( )** method description.
- In the *filePath* parameter, pass a string containing the path to the file that has the code to call for this handler. You can pass either an absolute path or a path relative to the project folder (POSIX syntax).
- In the *socketID* parameter, pass a local name for the WebSocket. This name is only used with the **removeWebSocketHandler( )** method.
- In the shared parameter, pass a Boolean value indicating if you want the local WebSocket to be handled through a **SharedWorker( )** or a **dedicated Worker( )**:
    - pass **true** to use a shared worker (recommended)
    - pass **false** to use a dedicated worker

  Using a shared worker is usually recommended because a single thread will be created for all WebSocket connections. It is also more appropriate if you want to share information between the clients.
  If you use a dedicated worker, each new client WebSocket connection will open a new thread on the server. In this case, you have to pay attention to the server memory depending on the number of concurrent connections.

Example

You want to install a WebSocket handler that will manage a chat service in your application. In the bootstrap file of your application, you write:

```
httpServer.addWebSocketHandler("/chat", "chat-server.js", "myChat", true);
    //"/chat" is the incoming WebSocket URL
    // "chat-server.js" is the script file located at the root of your project folder
    // "myChat" is the local name of the WebSocket
    // true means you want to use a shared worker
```

removeWebSocketHandler( )

void **removeWebSocketHandler**(String *socketID*)

Parameter	Type	Description
socketID	String	Local name of the WebSocket to remove

Description

The **removeWebSocketHandler( )** method removes the WebSocket handler *socketID* from the server.

WebSocket handlers are installed with the **addWebSocketHandler( )** method.

Example

You want to uninstall the "myChat" WebSocket:

```
httpServer.removeWebSocketHandler("myChat");
```