Access a SocketPro Server Anywhere by HTTP/WebSocket Protocols

Contents:

1. Introduction

SocketPro comes with system client and server core libraries which run on major platforms such as Windows (including Wince and Pocket PC) and Linux as well as Mac in the future. However, SocketPro core libraries, in the near future, will not be ported on various non-major platforms such as IBM mainframe, HP-Unix systems, Sun micro systems, etc. In addition, SocketPro already has a limited number of adapters available for major development environments. However, as you might conceive, SocketPro will not come with adapters for non-major development languages soon, although UDAParts is indeed trying innovative strategies to create new adapters for those languages with high market usage shares.

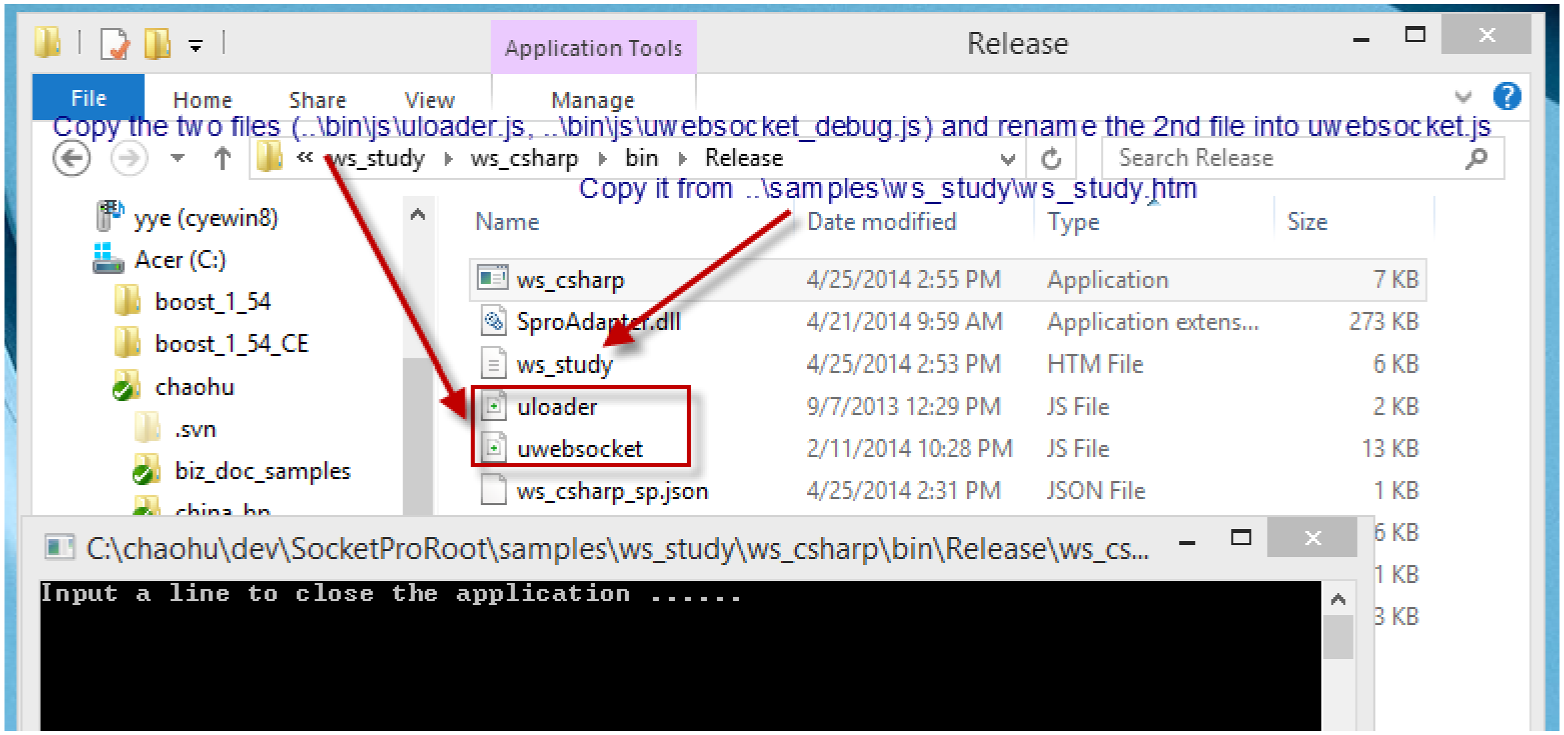
To solve the issues listed above, SocketPro server intentionally plans to create to a full-support web socket that allows the users in the previous situations access a SocketPro server application as long as it is enabled by HTTP/websocket service. We recommend that you use websocket protocol instead of classic HTTP protocol because websocket shows numerous advantages over traditional HTTP protocol, though SocketPro does support HTTP service as well.

Therefore, we create this short article to easily guide and quickly allow you to access a SocketPro server application from minor platforms or development languages by use of websocket protocol.

2. Easy way to learn SocketPro adapter for websocket

Compile C++ or C# project: To ease your use of websocket when accessing a SocketPro server application enabled with HTTP/websocket service, we create C++ and C# projects so that you can use one of two executable applications to experiment the supported requests. You could use any of several browsers as a client, which should have both debug console and websocket features supported. The projects are located at the directories *../socketpro/samples/ws_study/ws_cplusplus* and *../socketpro/samples/ws_study/ws_csharp*, respectively for C++ and C#.

SocketPro HTTP/websocket-enabled server settings: Afterwards, prepare a set of required files as shown in Figure 1 below. Note that you should use uwebsocket_debug.js instead of uwebsocket.js so that you can clearly watch how SocketPro javascript adapter works inside a browser. The second version of SocketPro adapater for websocket is given intentionally for your purposes, which could be helpful when writing your own adapter as you can easily step through code execution from a browser debugger.

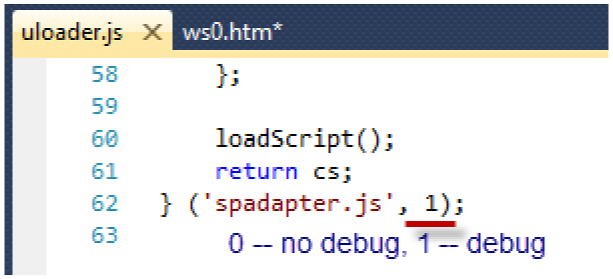

Application Tools
Release
File
Home
Share
View
Manage
Copy the two files (..\bin\js\uloader.js, ..\bin\js\uwebsocket_debug.js) and rename the 2nd file into uwebsocket.js
« ws_study ▸ ws_csharp ▸ bin ▸ Release
Search Release
Copy it from ..\samples\ws_study\ws_study.htm
yye (cyewin8)
Acer (C:)
boost_1_54
boost_1_54_CE
chaohu
.svn
biz_doc_samples
china_hn

Name	Date modified	Type	Size
ws_csharp	4/25/2014 2:55 PM	Application	7 KB
SproAdapter.dll	4/21/2014 9:59 AM	Application extens...	273 KB
ws_study	4/25/2014 2:53 PM	HTM File	6 KB
uloader	9/7/2013 12:29 PM	JS File	2 KB
uwebsocket	2/11/2014 10:28 PM	JS File	13 KB
ws_csharp_sp.json	4/25/2014 2:31 PM	JSON File	1 KB

```
C:\chaohu\dev\SocketProRoot\samples\ws_study\ws_csharp\bin\Release\ws_cs...
Input a line to close the application ......
```

Figure 1: Settings for a sample C# SocketPro server enabled with HTTP/websocket service

Enable JavaScript debug: At last, it is suggested that you turn on JavaScript adapter debug information inside the file uloader.js as shown in Figure 2.

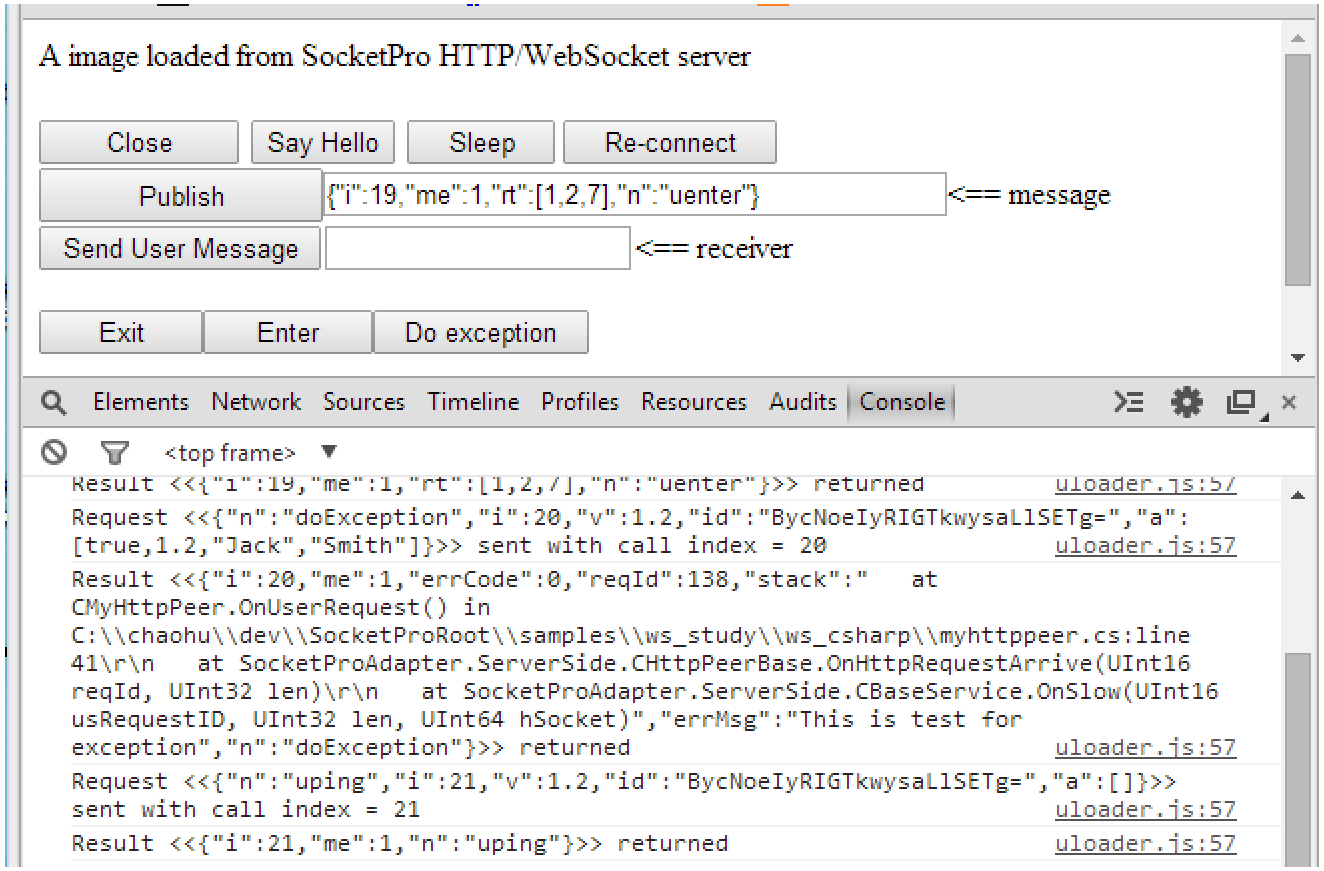


```
        };

        loadScript();
        return cs;
    } ('spadapter.js', 1);
```

Figure 2: Turn on or off SocketPro JavaScript adapter debug log

Once you turn on adapter debug log, you can see logged transaction JSON strings as shown in Figure 3 below.

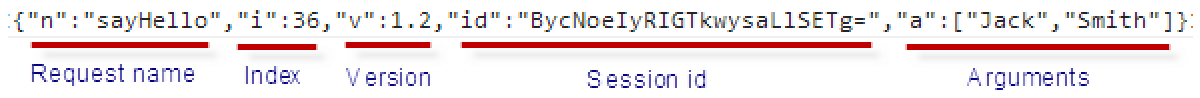

A image loaded from SocketPro HTTP/WebSocket server

```
Result <<{"i":19,"me":1,"rt":[1,2,7],"n":"uenter"}>> returned              uloader.js:57
Request <<{"n":"doException","i":20,"v":1.2,"id":"BycNoeIyRIGTkwysaL1SETg=","a":
[true,1.2,"Jack","Smith"]}>> sent with call index = 20                     uloader.js:57
Result <<{"i":20,"me":1,"errCode":0,"reqId":138,"stack":"   at
CMyHttpPeer.OnUserRequest() in
C:\\chaohu\\dev\\SocketProRoot\\samples\\ws_study\\ws_csharp\\myhttppeer.cs:line
41\r\n   at SocketProAdapter.ServerSide.CHttpPeerBase.OnHttpRequestArrive(UInt16
reqId, UInt32 len)\r\n   at SocketProAdapter.ServerSide.CBaseService.OnSlow(UInt16
usRequestID, UInt32 len, UInt64 hSocket)","errMsg":"This is test for
exception","n":"doException"}>> returned                                   uloader.js:57
Request <<{"n":"uping","i":21,"v":1.2,"id":"BycNoeIyRIGTkwysaL1SETg=","a":[]}>>
sent with call index = 21                                                  uloader.js:57
Result <<{"i":21,"me":1,"n":"uping"}>> returned                            uloader.js:57
```

Figure 3: Sample logged transaction JSON strings from SocketPro JavaScript adapter

3. Request JSON object

In short, SocketPro server side requires that you use JSON object with the following simple format, shown in Figure 4.



```
{"n":"sayHello","i":36,"v":1.2,"id":"BycNoeIyRIGTkwysaLlSETg=","a":["Jack","Smith"]}
```

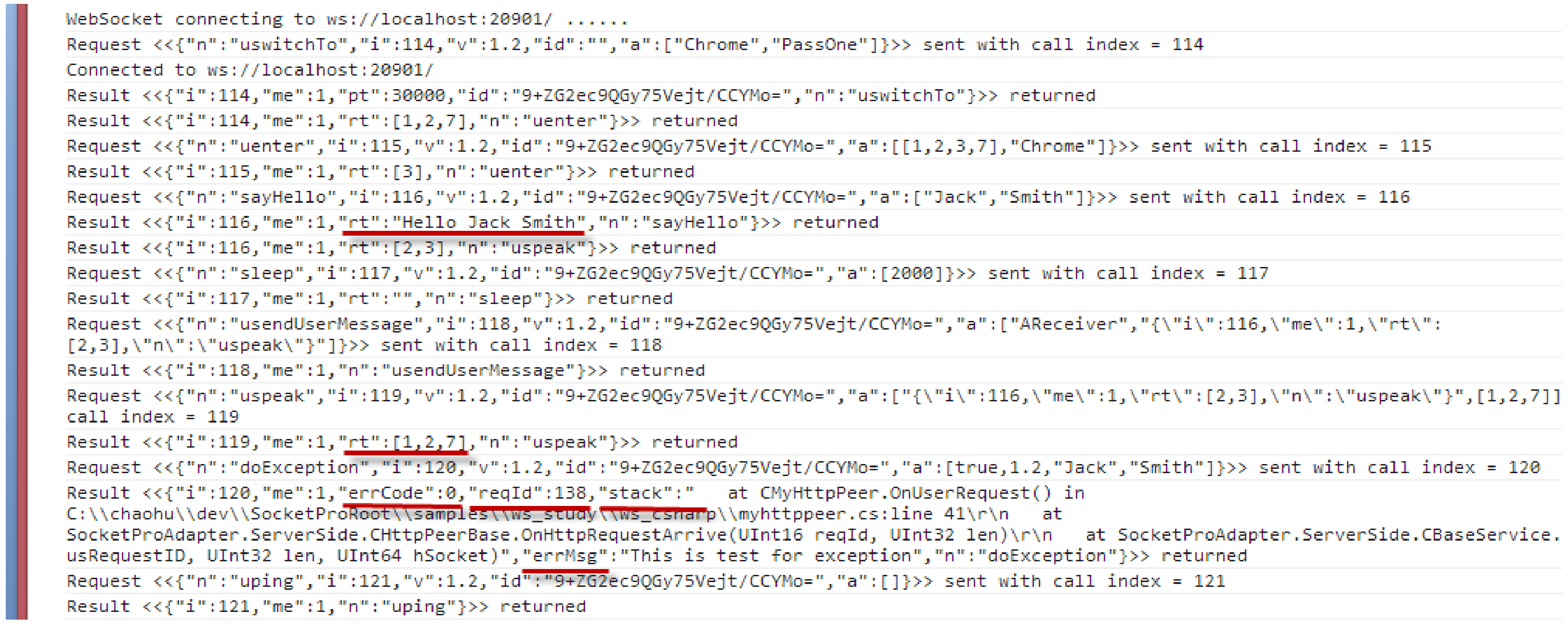
Figure 4: SocketPro JSON request object format

Note that the request index should always be increasing uniquely after web socket session is established. A session id is required for all requests except the request *uswitchTo* since its session id could be an empty string with zero length. The unique session id is automatically created at server side when a SocketPro server gets the request *uswicthTo*. The arguments array could be zero in length. Each of the arguments could be any type of valid JSON objects. SocketPro will use request name as a key to find a processing method at server side. The version is required, though its value is arbitrarily a two-digit number.

In regards to the details for each of the requests, you can easily access them from logged transaction JSON strings as shown in Figure 3 previously.

4. Response JSON object

Three basic members: A SocketPro server application will return a JSON object as a result with a limited number of variant formats as shown in Figure 5 below. However, all response JSON objects will contain three values, *i*, *me* and *n* for request index, self and request name, respectively.



```
WebSocket connecting to ws://localhost:20901/ ......
Request <<{"n":"uswitchTo","i":114,"v":1.2,"id":"","a":["Chrome","PassOne"]}>> sent with call index = 114
Connected to ws://localhost:20901/
Result <<{"i":114,"me":1,"pt":30000,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","n":"uswitchTo"}>> returned
Result <<{"i":114,"me":1,"rt":[1,2,7],"n":"uenter"}>> returned
Request <<{"n":"uenter","i":115,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":[[1,2,3,7],"Chrome"]}>> sent with call index = 115
Result <<{"i":115,"me":1,"rt":[3],"n":"uenter"}>> returned
Request <<{"n":"sayHello","i":116,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":["Jack","Smith"]}>> sent with call index = 116
Result <<{"i":116,"me":1,"rt":"Hello Jack Smith","n":"sayHello"}>> returned
Result <<{"i":116,"me":1,"rt":[2,3],"n":"uspeak"}>> returned
Request <<{"n":"sleep","i":117,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":[2000]}>> sent with call index = 117
Result <<{"i":117,"me":1,"rt":"","n":"sleep"}>> returned
Request <<{"n":"usendUserMessage","i":118,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":["AReceiver","{\"i\":116,\"me\":1,\"rt\":
[2,3],\"n\":\"uspeak\"}"]}>> sent with call index = 118
Result <<{"i":118,"me":1,"n":"usendUserMessage"}>> returned
Request <<{"n":"uspeak","i":119,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":["{\"i\":116,\"me\":1,\"rt\":[2,3],\"n\":\"uspeak\"}",[1,2,7]]
call index = 119
Result <<{"i":119,"me":1,"rt":[1,2,7],"n":"uspeak"}>> returned
Request <<{"n":"doException","i":120,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":[true,1.2,"Jack","Smith"]}>> sent with call index = 120
Result <<{"i":120,"me":1,"errCode":0,"reqId":138,"stack":"   at CMyHttpPeer.OnUserRequest() in
C:\\chaohu\\dev\\SocketProRoot\\samples\\ws_study\\ws_csharp\\myhttppeer.cs:line 41\r\n   at
SocketProAdapter.ServerSide.CHttpPeerBase.OnHttpRequestArrive(UInt16 reqId, UInt32 len)\r\n   at SocketProAdapter.ServerSide.CBaseService.
usRequestID, UInt32 len, UInt64 hSocket)","errMsg":"This is test for exception","n":"doException"}>> returned
Request <<{"n":"uping","i":121,"v":1.2,"id":"9+ZG2ec9QGy75Vejt/CCYMo=","a":[]}>> sent with call index = 121
Result <<{"i":121,"me":1,"n":"uping"}>> returned
```

Figure 5: Sample response JSON objects from SocketPro server with highlights

uswitchTo members: As shown in Figure 5 above, the request *uswitchTo* will return with two extra members, *pt* and *id*, which represents expected heart beat interval time in milliseconds and an auto-generated unique session identification number which must be used by all other requests.

User defined returning string result: As shown in the sample requests *sayHello* and *sleep*, a user defined request must return a string (*rt*) except in the case where there is an exception in processing.

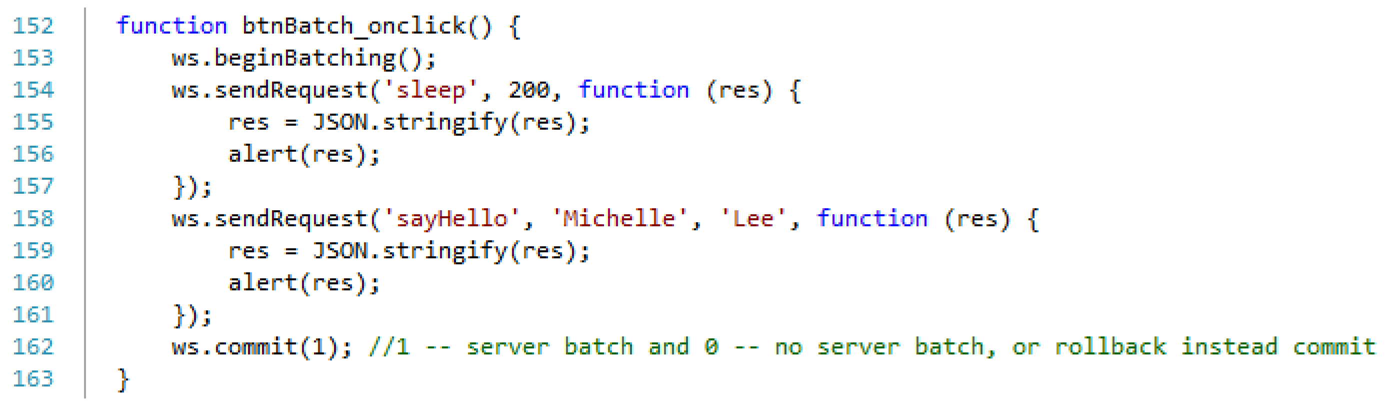
Exception members: If there is an exception caught at server side, SocketPro will return extra four members, *errCode*, *reqId*, *stack* and *errMsg*, as shown in the request *doException* in Figure 5 above.

Chat-related request returning strings: In regards to the three requests *uenter*, *uspeak* and *uexit*, their return results are an array of integers, which represents chat group identification numbers.

Caught chat-related messages: In regards to the caught chat related messages, these responses will have four members, *sender*, *serviceId*, *groups* and *msg* that correlate in order with each of the following listed: message source sender, sender service identification number, sender chat group identification number and message. Note the member *me* should be zero to indicate that the message does not originate from my-self.

5. Batch request JSON object

As you may already know, SocketPro supports request batch for the best network efficiency. However, it is optional if you wish to do so as is shown in Figure 6 below.



```
function btnBatch_onclick() {
    ws.beginBatching();
    ws.sendRequest('sleep', 200, function (res) {
        res = JSON.stringify(res);
        alert(res);
    });
    ws.sendRequest('sayHello', 'Michelle', 'Lee', function (res) {
        res = JSON.stringify(res);
        alert(res);
    });
    ws.commit(1); //1 -- server batch and 0 -- no server batch, or rollback instead commit
}
```

Figure 6: Send multiple requests in batch by help of SocketPro adapter for JavaScript

Sending requests in batch will create a following JSON request object string.

{"n":"udoBatch","i":4,"v":1.2,"id":"f/5QS3A3SgGLlRckPbLmP9k=","a":[true,[{"n":"sleep","i":2,"v":1.2,"id":"f/5QS3A3SgGLlRckPbLmP9k=","a":[200]},{"n":"sayHello","i":3,"v":1.2,"id":"f/5QS3A3SgGLlRckPbLmP9k=","a":["Michelle","Lee"]}]]}

As you can see, the two requests above in Figure 6 are embedded into the batch request argument array.

6. Requests to be implemented

If you have a good background in JavaScript, you could easily and quickly convert our JavaScript adapter into yours. The following table lists all required requests that could be implemented.

Request name	Comments
Generic request	Base request reused to create other requests
uping	It is served as heart-beat. Adapter internally calls it automatically at an interval time as shown in the section 4 from the request *uswitchTo*
uswitchTo	Calling the method with user id and password to ask for SocketPro HTTP/websocket service
uenter (optional)	Calling the method from client to subscribe chat topics. It is optional as you can subscribe topics at server side
uexit (optional)	Calling the method from client to unsubscribe chat topics. It is optional as you can unsubscribe topics at server side
uspeak (optional)	Calling the method from client to publish a message onto one or more chat groups or topics. It is optional as you can publish a message at server side
usendUserMessage (optional)	Calling the method from client to send a message onto a receiver. It is optional as you can send a receiver a message at server side
udoBatch (optional)	Implementing this method enables requests batch at client for better network efficient. It is optional but highly recommended

Table 1: A list of requests to be implemented from your adapter

7. Error codes and their error messages

The following table 2 lists all possible error codes and their error messages from SocketPro HTTP/websocket service.

Error code	Error message
0	Ok
1	Bad JSON request
2	Bad number of arguments
3	Bad arguments
4	Bad argument data type
5	Bad request

6	Empty request
7	Not supported
8	Authentication failed

Table 2: A list of error codes and their error messages

If an error is detected at server side, its response JSON object will be added with a new member *rc*, and it will be set to a proper integer error code as shown in table 2 above.