xockets.io

(XPages WebSockets setup guide)

1. Download the xockets.io zip file
2. To just install the websocket dependencies import only the com.tc.websocket.updatesite into an update site database on your Domino / Xwork server
3. If you want to use TLS for secure network connections create a keystore.jks file from the command line *(keytool -genkey -validity 3650 -keystore "keystore.jks" -storepass "storepassword" -keypass "keypassword" -alias "default" -dname "CN=127.0.0.1, OU=MyOrgUnit, O=MyOrg, L=MyCity, S=MyRegion, C=MyCountry")*
4. Store the keystore.jks file on the server
5. Drop the chat.nsf on your server (sign and set anonymous to no access, default to author)
   - see chat.js javascript for simple websocket client example.
   - See fmchat for class notes web form chat client example
   - see chat.xsp for xpage example
6. Drop websocket.nsf on the root of your server (sign and set anonymous no access, default to author)
   - For the Broadcast Stock Data agent you will need to turn on the web task from the console (load web) should do it.
7. Set java.policy file to allow all permissions (see below for sample working java.policy file)

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};


// default permissions granted to all domains

grant {
        permission java.security.AllPermission;
};

// Notes java code gets all permissions

grant codeBase "file:${notes.binary}/*" {
        permission java.security.AllPermission;
};

grant codeBase "file:${notes.binary}/rjext/*" {
        permission java.security.AllPermission;
};

grant codeBase "file:${notes.binary}/ndext/*" {
        permission java.security.AllPermission;
};
```

```
grant codeBase "file:${notes.binary}/xsp/-" {
        permission java.security.AllPermission;
};


grant codeBase "file:${notes.binary}/osgi/-" {
        permission java.security.AllPermission;
};
```

8) NOTE: The notes.ini settings documented below can now be stored / referenced from a profile document in websocket.nsf (profile document settings will take precedence over notes.ini) (see actions menu in websocket.nsf)

```
Config Data:
#start websocket settings
WEBSOCKET_PORT=8889
#debug/test settings
WEBSOCKET_DEBUG=false
WEBSOCKET_TEST_MODE=true
WEBSOCKET_MAX_MSG_SIZE=15977336
#security settings
#WEBSOCKET_FILTER=com.tc.websocket.filter,com.tc.websocket.filter.AllowedCharsFilter
WEBSOCKET_ALLOW_ANONYMOUS=true
WEBSOCKET_THREAD_COUNT=1
WEBSOCKET_EVENT_LOOP_THREADS=1
WEBSOCKET_MAX_CONNECTIONS=20005
WEBSOCKET_ALLOWED_ORIGINS=*
#network encryption settings
WEBSOCKET_ENCRYPT=false
WEBSOCKET_KEYSTORE_PATH=C:/websocket-certs/websocket2.jks
WEBSOCKET_KEY_PASSWORD=xxxxxxx
WEBSOCKET_KEYSTORE_PASSWORD=xxxxxxx
WEBSOCKET_KEYSTORE_TYPE=JKS
```

(add a new line between each key/value pair)

Readers:
᠇centosdev/marksdev
mambler-mac-win/marksdev ᠇ ▼

(make sure the server can read this document)

9) The following settings can be applied in the Config profile document of the websocket.nsf, or the notes.ini

- WEBSOCKET_PORT=8889
  - port you want to run the websocket server on.

- WEBSOCKET_MONITOR_QUEUE=true
  - if the above is set to true, the threads that monitor the different queuing views in the websocket.nsf will be turned on.
- WEBSOCKET_TEST_MODE=false
  - if the above is true, it opens up the DominoWebSocket server to direct testing (see project com.tc.websocket.tests).
- WEBSOCKET_MAX_MSG_SIZE defaults to 1048576 bytes
  - Maximum size for a json message. Unlimited size is no longer supported.
- WEBSOCKET_MAX_CONNECTIONS defaults to 200
  - Maximum number of websocket connections allowed on a server. Once max is reached new websocket connections will be closed immediately.
- WEBSOCKET_ALLOWED_ORIGINS (comma delimited string) must be set to your allowed hosts (e.g. yoursite.com,yourothersite,com,mysite.com)
  - Security measure to check the Origin http header of incoming requests. If the Origin doesn't match the allowed list, websocket connection will not be allowed.
  - If an asterisk is used, allows "ALL" origins (not recommended)
- WEBSOCKET_CLUSTERED defaults to false
  - tells the plugin to turn on monitoring background threads
- WEBSOCKET_BROADCAST_SERVER=CN=mambler-mac-win/O=marksdev
  - denotes the only server in the cluster that should process broadcast messages.
- WEBSOCKET_CLUSTERMATE_MONITOR=CN=centosdev/O=marksdev
  - in the above, the current server monitors the status of centosdev
- WEBSOCKET_CLUSTERMATE_EXPIRATION=120
  - tells the ClustermateMonitor when to start dropping users to offline status after N seconds
- WEBSOCKET_PURGE_INTERVAL defaults to 900 sec
  - instead of relying on agents to cleanup expired messages and users

Security Related Settings
- WEBSOCKET_ENCRYPT=true
  - tells the plugin that you want to use TLS encryption over the network
- WEBSOCKET_ALLOW_ANONYMOUS=false
  - does not allow anonymous connections. Messages received by anonymous will be ignored, and connections dropped
- WEBSOCKET_KEYSTORE_PATH=C:\websocket-certs\keystore.jks
  - if WEBSOCKET_ENCRYPT=true the above must be setup
- WEBSOCKET_KEY_PASSWORD=keypassword
  - if WEBSOCKET_ENCRYPT=true the above must be setup
- WEBSOCKET_KEYSTORE_PASSWORD=storepassword
  - if WEBSOCKET_ENCRYPT=true the above must be setup
- WEBSOCKET_KEYSTORE_TYPE=JKS
  - if WEBSOCKET_ENCRYPT=true the above must be setup

- WEBSOCKET_USER (must be populated)
  - All Domiono sessions created by the xockets.io server will use this Id
- WEBSOCKET_PASSWORD (must be populated)
  - Password used for the WEBSOCKET_USER
- WEBSOCKET_FILTER=com.tc.websocket.filter,com.tc.websocket.filter.AllowedCharsFilter
  - the above param allows for custom data filters on SocketMessage objects. See the above for an example

Performance Related Settings (adjust the below according to your needs and server specs)
- WEBSOCKET_THREAD_COUNT defaults to 1
  - used for the background operations (e.g. processing queues, user cleanup, purging old data etc)
- WEBSOCKET_EVENT_LOOP_THREADS defaults to 1
  - Used to manage the event loop that processes incoming and outgoing websocket messages.
- WEBSOCKET_SEND_BUFFER defaults to 16384 bytes
  - Buffer for outbound websocket messages
- WEBSOCKET_RECEIVE_BUFFER defaults to 16384
  - Buffer for inbound websocket messages
- WEBSOCKET_COMPRESSION_ENABLED defaults to false
  - If supported by the user agent (browser / client), the server will attempt to compress the websocket message
  - Note: during testing when compression was turned on it seemed to consume more memory than off. Depending on your message size you may want to consider keeping this turned off.

example notes.ini params:
#this setting is exremely important to ensure proper loading of the websocket plugin.
XPagesPreload=1
XPagesPreloadDB=websocket.nsf

Below are my current settings for localhost/dev (can be dropped into profile, or notes.ini)

#start websocket settings
WEBSOCKET_PORT=8889
#username used for server-side operations (e.g. agent signer)
WEBSOCKET_USER=admin
WEBSOCKET_PASSWORD=xxxxxx
#debug/test settings
WEBSOCKET_DEBUG=false
WEBSOCKET_TEST_MODE=true
WEBSOCKET_MAX_MSG_SIZE=15977336
#security settings

#WEBSOCKET_FILTER=com.tc.websocket.filter,com.tc.websocket.filter.AllowedCharsFilter

WEBSOCKET_ALLOW_ANONYMOUS=true

WEBSOCKET_THREAD_COUNT=1

WEBSOCKET_EVENT_LOOP_THREADS=2

WEBSOCKET_MAX_CONNECTIONS=8000

WEBSOCKET_ALLOWED_ORIGINS=*

#network encryption settings

WEBSOCKET_ENCRYPT=false

WEBSOCKET_KEYSTORE_PATH=C:/websocket-certs/websocket2.jks

WEBSOCKET_KEY_PASSWORD=xxxxxx

WEBSOCKET_KEYSTORE_PASSWORD=xxxxxx

WEBSOCKET_KEYSTORE_TYPE=JKS

WEBSOCKET_COMPRESSION_ENABLED=false

WEBSOCKET_SEND_BUFFER=16384

WEBSOCKET_RECEIVE_BUFFER=16384

Available Command Line Operations:

- tell http osgi websocket stop   (Run this prior to terminating http to avoid server crashes. This command will terminate the threads cleanly)
- tell http osgi websocket start
- tell http osgi websocket count   (gets the current count of websockets on the current server)
- tell http osgi websocket count-all   (gets the count of all the websockets across a cluster)
- tell http osgi websocket show-all-users   (shows all users currently using a websocket across a cluster)
- tell http osgi websocket show-users   (shows users on current server)

  ○

Other notes about security and the browsers:

If you are using a self-signed cert (i.e. using process from above), be sure to copy and paste the websocket url into the address bar of the browser, and alter the wss:// to https://, and hit enter. That will force the browser to prompt you to add an exception for the cert. (i.e. **wss://**mambler-mac-win:8889/DD469405D2D13674C7AE2C0D5BCB9662DE57D84F to **https://**mambler-mac-win:8889/DD469405D2D13674C7AE2C0D5BCB9662DE57D84F). With Chrome I noticed that I had to do this every time I restarted the browser. FireFox persisted the exception. After doing this you may need to restart http on the domino server, then refresh the browser to chat.nsf url.

Setup XPage / Domino WebSockets with a valid Certificate

Recently, I was approached by another XPager to help setup TLS / SSL using a valid certificate (preferably re-use the same cert as the Domino https server) for XPage WebSockets. As usual, something that seemed "easy" took a bit longer than expected to research, cycle through, and solve. The below set of steps assumes you have the private key, issued certificate, and the CA's root and intermediate certificate that you are already using on your Domino server, and that you have OpenSSL, and Java installed.

1) Issue an openssl command to convert the certs into a .p12 store (make sure you start cmd prompt as admin, and record the password you provide)

openssl pkcs12 -export -name {youralias} -in {yourcertificate} -inkey {yourprivatekey} -out {yourp12file}

2) Convert PKCS12 keystore into a JKS keystore

keytool -importkeystore -destkeystore {yourkeystore.jks} -srckeystore {yourp12file} -srcstoretype pkcs12 -alias {youralias}

3) Import the root, and intermediate certificates (run command once per file)

keytool -import -alias {use-adifferent-alias-for-each-file} -file {certfile} -keystore {yourkeystore.jks}

4) Drop the .jks file into the target directory referenced in your server's notes.ini

5) Make sure the notes.ini is setup to use encryption over the wire (see below)

WEBSOCKET_ENCRYPT=true
WEBSOCKET_KEYSTORE_PATH=C:\websocket-certs\websocket.jks
WEBSOCKET_KEY_PASSWORD=*******
WEBSOCKET_KEYSTORE_PASSWORD=*******
WEBSOCKET_KEYSTORE_TYPE=JKS

Below are the commands I used to setup my local server. I used StartSSL's free certificates when setting this up.

openssl pkcs12 -export -name tekcounsel -in 2_tekcounsel.net.crt -inkey private.key -out keystore.p12

keytool -importkeystore -destkeystore websocket.jks -srckeystore keystore.p12 -srcstoretype pkcs12 -alias tekcounsel

keytool -import -alias websocket_cert -file {certfile} -keystore websocket.jks

Scale:
During local testing the server scaled up to 15000 websocket connections without any issues so long as the messages were small (around 300kb), and adequate memory was allocated to the heap (8192M). Most of the "normal" testing was used with around 8000 users / websocket connections with a heap of 4096M. Scale has been dramatically improved by using the Netty libraries since the prior release.

Browser Info:
Load up the chat application in two separate browsers that support websockets. I did local testing with chrome Version 33.0.1750.154 m, and FireFox 27.0

REST APIs (see chat.ntf classic notes web form example, add websocket to the Domino Access Services)

POST request to register user (responds with a list of all on line users, and the websocketurl)

/api/websocket/v1/registeruser

POST request to undo user registration (responds with simple success code)

/api/websocket/v1/removeuser

GET request to pull the online users:

/api/websocket/v1/onlineusers

GET request to pull the correct websocketurl:

/api/websocket/v1/websocketurl

POST request to send a socket message to an online user:

/api/websocket/v1/sendmessage

GET request to send a simple websocket message:

/api/websocket/v1/sendsimple?from={from}&to={to}&text={text}

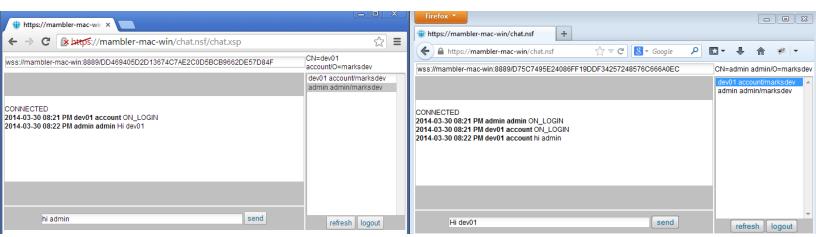**GET request to pull in the latest message for a user (user cannot have an open websocket connection)**

/api/websocket/v1/latestmessage

GET request to pull in all the pending messages for a user (user cannot have an open websocket connection)

/api/websocket/v1/messages

Other info about REST API (check out the below)
- com.tc.websocket.tests
- Domino wiki on how to setup REST services on domino,
- websocketbean.js in chat.ntf

Screen shot of working sample:

RequestURI Routing Path:

 send targeted websocket messages to users on a specific page to render system / log data on a dashboard to monitor the client state. I don't want users getting socket messages in an app that is expecting a different type of socket message data. As a result of this need, I've gone ahead and updated the Domino Websocket plugin to include support for routing messages based on the user's request URI and role. Below are a few examples of the routing that can be achieved with a simple string in the "to" attribute of the socket message from any of your websocket enabled applications. See the chat.ntf application for a working copy of the samples below.

/chat.nsf*[mgr]   (sends to everyone in the chat application with [mgr] role)

/chat.nsf/chat.xsp[mgr],[admin] (sends to everyone on the specific chat.xsp page with the [mgr] or [admin] role)

/chat.nsf (sends to anyone on the default page of chat.nsf)

/chat.nsf* (sends to everyone on any page/xpage in chat.nsf)

/chat.nsf/chat.xsp/CN=admin admin/O=marksdev (send direct message to user admin admin on page chat.xsp)
The request URI is also stored with the user profile created when the websocket session is established, to allow for more complex querying by background agents, SSJS, or Java (see Users by URI view in websocket.ntf).
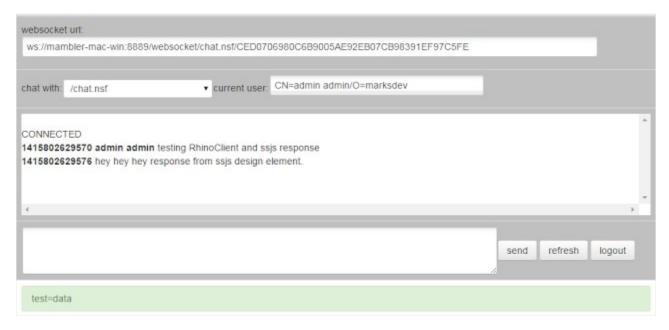
SSJS Websocket Client
developers will be able to register SSJS libraries to consume / intercept websocket client events via simple API call. **NOTE: The SSJS is compiled and executed under the Rhino scripting engine, not the XPages version of SSJS. There are some syntax differences you need to be aware of, namely type declarations (i.e. var str:String = "some string" will not compile must only be var str="some string").** Be sure to omit them when writing SSJS for websockets. Below are the steps required to register a script library with the websocket plugin:

   1) Create your SSJS script library websocket client (must be of type SSJS when creating lib in designer)

```
//init code block start
if("onOpen".equals(event)){
    onOpen();
}else if("onMessage".equals(event)){
    onMessage();
}else if("onClose".equals(event)){
    onClose();
}else if("onError".equals(event)){
    onError();
}
//init code block end

function onOpen(){
    print("onOpen event for message " + handShake + " on platform " + session.getPlatform());
}
function onClose(){
    print("onClose event on platform " + session.getPlatform());
}
function onError(){
    print("onError event for message " +  ex.getMessage() + " on platform " + session.getPlatform());
}
function onMessage(){
    print("onMessage event for message " + socketMessage.getText() + " on platform " + session.getPlatform());

    //log some data to the chat.nsf using some notes backend objects.
    var db = session.getDatabase("","chat.nsf");
    var doc = db.createDocument();
    doc.replaceItemValue("Form","fmLog");
    var rtitem = doc.createRichTextItem("Body");
    rtitem.appendText(socketMessage.getJson());
    doc.save();

    //run an agent...
    var agent = db.getAgent("SampleAgent");
    agent.run(doc.getNoteID());


    //respond to the same requestURI channel
    var msg = websocketClient.createMessage();
    msg.setTo("/chat.nsf"); //respond to all on this uri / channel
    msg.setText("hey hey hey response from ssjs design element.");
    msg.getData().put("test","data");
    websocketClient.sendMsg(msg);
}
function print(str){
    //example on how to load a class from a loaded osgi bundle
    var printTest = bundleUtils.load("com.tc.utils", "com.tc.utils.PrintTest");
    printTest.print(str);
}
```

Below is the output from the SSJS client to any users on /chat.nsf

2) Register your Rhino compliant SSJS lib a few... via SSJS API call (username, password only supported from API)

Language: JavaScript (Server Side)

Condition:

```
if(!websocketBean.containsSocketListener("/chat.nsf")){
    websocketBean.addSocketEventListener("/chat.nsf","*","/chat.nsf/ssjs.websocket.listener");
}
```


Language: JavaScript (Server Side)

Condition:

```
if(!websocketBean.containsSocketListener("/chat.nsf/ssjs.websocket.listener")){
//you can now run the listeners as session than globally configured for websocket.
    websocketBean.addSocketEventListener("/chat.nsf","*","/chat.nsf/ssjs.websocket.listener", "chatapp","password");
}
```

**or
command line**

```
> tell http osgi websocket register-script localhost /chat.nsf * /chat.nsf/ssjs.websocket.listener
[0E18:14CF-0ED4] 11/10/2014 08:05:53 PM  HTTP JVM: 2014/11/10 20:05:53.533 INFO putting compiled scr
cript() ::thread=pool-2-thread-3 ::loggername=com.tc.websocket.clients.RhinoClient
[0E18:14D1-1890] 11/10/2014 08:05:53 PM  HTTP JVM: 2014/11/10 20:05:53.535 INFO executing onOpen scr
en() ::thread=Thread-58 ::loggername=com.tc.websocket.clients.RhinoClient
[0E18:14D2-1890] 11/10/2014 08:05:53 PM  HTTP JVM: onOpen event for message org.java_websocket.hands
>
```

(note that * covers all events onOpen, onMessage, onClose, and onError)

A few other important notes about the default objects in scope of your script(s):

- **session** (uses server's Id, so make sure the server has appropriate rights to the target nsf housing the script lib)
- **websocketClient** (you can use this object to send messages out in response to incoming events)
- **event** (just the name of the event onOpen, onClose, onMessage, onError)
- **socketMessage** is only available via onMessage event
- **bundleUtils** (facilitates accessing specific OSGi plugins in the XPage runtime see print method in above code, target class must use no arg constructor)

New Command line options:
- **register-script** (takes four arguments **host uri event scriptpath**)
- **reload-scripts** (takes no arguments and reloads from source nsf and re-compiles all scripts)
- **show-scripts** (takes no arguments, renders all the registered scripts)
- **remove-script** (takes one argument that is the path to the lib i.e. /chat.nsf/scriptlib)