Domino Websocket Setup

based on the efforts from http://java-websocket.org/

1. Download the webshell-xpages-ext-lib
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. Make sure your server's password is empty as the Config object needs to load the below settings from the notes.ini
9. The notes.ini settings are no longer required by default except for the pre-load items. Below is a list of the default settings (these can be overridden):

#these two settings are extremely important to ensure proper loading of the websocket plugin (needs to load prior to any .nsf application referencing the plugin directly)

XPagesPreload=1
XPagesPreloadDB=websocket.nsf

WEBSOCKET_PORT=8889
WEBSOCKET_MAX_CONNECTIONS=100
WEBSOCKET_MAX_MSG_SIZE=1048576 (about 1mb)
WEBSOCKET_ENCRYPT=false
WEBSOCKET_ALLOW_ANONYMOUS=false
WEBSOCKET_CLUSTERED=false
WEBSOCKET_FILTER=null

10. Apply the following notes.ini settings to override the above if needed (settings below assume encryption, bounce domino, eventually I'll move the passwords to a .nsf....):
    - WEBSOCKET_DEBUG=true
        - just sets the internal debug flag for the org_javawebsocket api
    - WEBSOCKET_ENCRYPT=true
        - tells the plugin that you want to use TLS encryption
    - WEBSOCKET_ALLOW_ANONYMOUS=false
        - does not allow anonymous connections. Messages received by anonymous will be ignored
    - WEBSOCKET_KEYSTORE_PATH=C:\websocket-certs\keystore.jks
        - if WEBSOCKET_ENCRYPT=true the above must be setup (see step 3)
    - WEBSOCKET_KEY_PASSWORD=keypassword
        - if WEBSOCKET_ENCRYPT=true the above must be setup (see step 3)
    - WEBSOCKET_KEYSTORE_PASSWORD=storepassword
        - if WEBSOCKET_ENCRYPT=true the above must be setup (see step 3)

- WEBSOCKET_KEYSTORE_TYPE=JKS
  - if WEBSOCKET_ENCRYPT=true the above must be setup (see step 3)
- 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_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
- 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 (in bytes)
  - Maximum size for a json message. If 0 any size is allowed
- WEBSOCKET_MAX_CONNECTIONS
  - Maximum number of websocket connections allowed on a server. Once max is reached new websocket connections will be closed immediately.
- WEBSOCKET_CLUSTERED=true
  - 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. **(MUST be present to send broadcast even if clustering is not setup)**
- 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_THREAD_COUNT defaults to 2
  - (this only impacts threads loaded for routing. Websocket workers are still equivalent to the number of cores / CPUs on the host machine)
- WEBSOCKET_ARRAY_QUEUE defaults to false
  - allows admin to change the internal data structure from LinkedBlockingQueue to ArrayBlockingQueue. No performance advantages were seen during testing. This was added to see if a bound blocking queue would reduce the memory foot print during JUnit test runs. Performance results were inconclusive
- WEBSOCKET_ARRAY_QUEUE_SIZE defaults to 10k
  - if WEB_SOCKET_ARRAY_QUEUE is set to true, uses this value to set the upper bound of

the array blocking queue.

- WEBSOCKET_PURGE_INTERVAL defaults to 900 sec
  - instead of relying on agents to cleanup expired messages and users

example notes.ini params:

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

#start websocket settings
WEBSOCKET_PORT=8889

#debug/test settings
WEBSOCKET_DEBUG=false

#set this to true to use the Junit test project
WEBSOCKET_TEST_MODE=false

#security settings
WEBSOCKET_FILTER=com.tc.websocket.filter,com.tc.websocket.filter.AllowedCharsFilter
WEBSOCKET_ALLOW_ANONYMOUS=false
WEBSOCKET_MAX_MSG_SIZE=0
WEBSOCKET_MAX_CONNECTIONS=100

#network encryption settings
WEBSOCKET_ENCRYPT=false
WEBSOCKET_KEYSTORE_PATH=C:\websocket-certs\keystore.jks
WEBSOCKET_KEY_PASSWORD=keypassword
WEBSOCKET_KEYSTORE_PASSWORD=storepassword
WEBSOCKET_KEYSTORE_TYPE=JKS

#clustered server settings
WEBSOCKET_CLUSTERED=true
WEBSOCKET_CLUSTERMATE_MONITOR=CN=familyroom/O=marksdev

WEBSOCKET_CLUSTERMATE_EXPIRATION=1200
WEBSOCKET_BROADCAST_SERVER=CN=mambler-mac-win/O=marksdev

#end websocket settings

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.

Scale:

During local testing the server scaled up to 2000 websocket connections without any issues as long as the time between connections was at least 500 milliseconds. This may have been due to having both the test client and websocket server on the same machine. When set to 0 time between connections the server seemed to manage OK with immediate connections of 20 users before throwing up connection refused exceptions. Again, this maybe due to having both clients and server on the same machine. See the following blogpost for more information about benchmarks:

http://markwambler.blogspot.com/2014/12/domino-websocket-fixes-performance.html

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

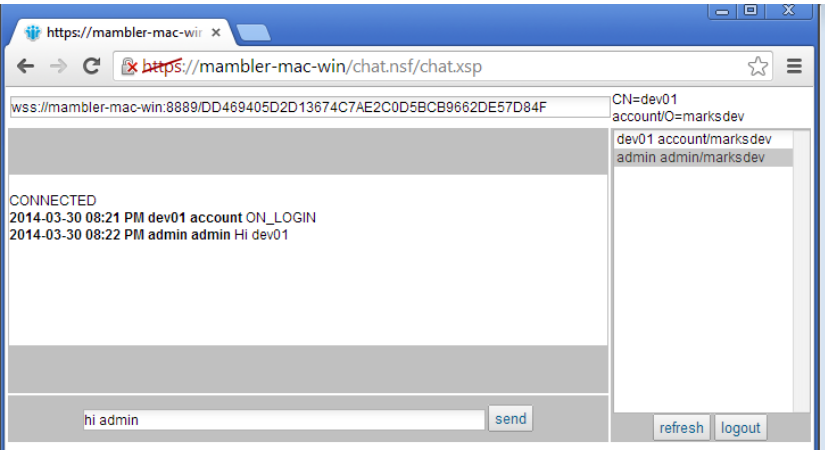
Other info about REST API (check out the below)

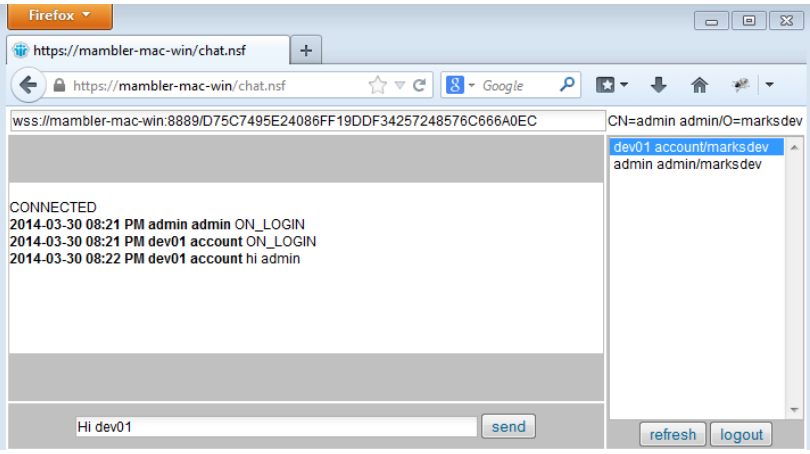
- com.tc.websocket.tests
- Domino wiki on how to setup REST services on domino,
- websocketbean.js in chat.ntf

Other Notes:

- The com.tc.websocket ext lib uses a session listener to load map the user's userId to their sessionId.
- The only security measures are now the WEBSOCKET_ALLOW_ANONYMOUS notes.ini param, and the WEBSOCKET_FILTER that can be applied to the SocketMessage
- Be sure to click the logout button when switching between users on the same browser to make sure your info is dropped from the DominoWebSocketServer.
- Was only tested on my local server running Windows 7. Not sure how it will behave on a different OS.

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.

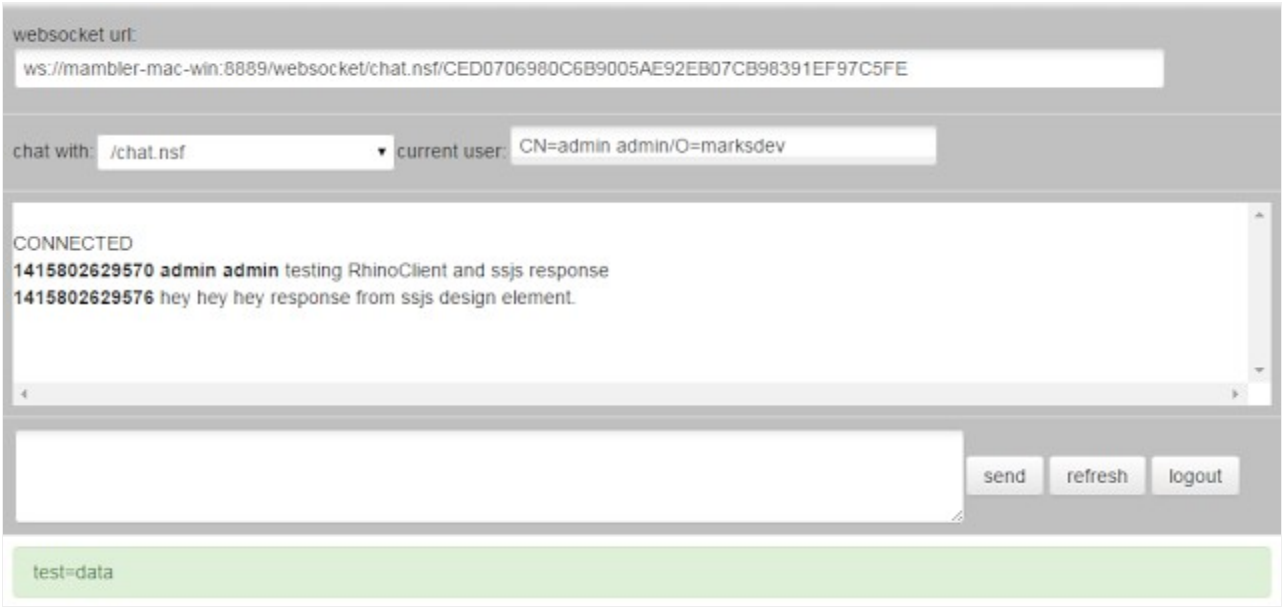
NOTE: The SSJS is compiled and executed under the Rhino scripting engine, not the XPages version of SSJS. There are some syntax differences you need to be aware of, namely type declarations (i.e. var str:String = "some string" will not compile must only be var str="some string"). Be sure to omit them 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 couple of ways... via SSJS API call

or
command line

(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)