Create a WebSocket server with the Wowza Streaming Engine Java API

With Wowza Streaming Engine™ media server software version 4.5.0 and later, you can use an HTTP provider to host a WebSocket communication session. HTTP providers are Java classes that are configured on a per-virtual host basis. They are lightweight web applications that can send information to or obtain information from a Wowza Streaming Engine server instance. This article describes how to create an HTTP provider to host a WebSocket session.

A WebSocket session provides a persistent connection between a browser and Wowza Streaming Engine. The WebSocket protocol can be used to send text or binary data between browsers. Examples of WebSocket applications are text, chat, out-of-band metadata, and control data.

**Note:** Wowza Streaming Engine 4.5.0 is required for HTTP providers to support WebSocket sessions.

**Introduction**

HTTP providers are lightweight HTTP web applications that extend Wowza Streaming Engine functionality. They are configured on a per-port basis in `[install-dir]/conf/VHost.xml`. An individual HTTP provider can be protected by a user name and password. Multiple HTTP providers can be attached to a single port and a specific HTTP provider can be selected based on a request filter. HTTP providers can also host WebSocket sessions for sending asynchronous, bi-directional information to and from a Wowza Streaming Engine instance.

The WebSocket protocol provides full-duplex communication changes over a TCP connection. The WebSocket protocol is supported by most modern web browsers and can be used with socket servers such as node.js. A WebSocket connection starts as an HTTP upgrade request. In your HTTP provider, you can accept the upgrade request and a WebSocket session is created and used to send and receive messages.

**Prerequisites**
You’ll need the Wowza™ IDE for Eclipse. See Extend Wowza Streaming Engine using the Wowza IDE.

Create an HTTP provider to host WebSocket sessions

Each HTTP provider extends the HTTPProvider2Base. For more information about creating and configuring HTTP providers, see Create a Wowza Streaming Engine HTTP provider.

The following is an example of an HTTP provider that supports the WebSocket protocol:

```java
import java.util.*;
import org.apache.commons.lang.time.*;
import com.wowza.util.*;
import com.wowza.wms.http.*;
import com.wowza.wms.logging.*;
import com.wowza.wms.server.*;
import com.wowza.wms.util.*;
import com.wowza.wms.vhost.*;
import com.wowza.wms.websocket.model.*;

public class HTTPProviderWebSocket extends HTTPProvider2Base {
  private static final Class CLASS = HTTPProviderWebSocket.class;
  private static final String CLASSNAME = "HTTPProviderWebSocket";
  public static final String DATEFORMAT = "EEE, dd MMM yyyy HH:mm:ss";
  public static final int TIMER_INTERVAL = 6000;
  private FastDateFormat fastDateFormat = FastDateFormat.getInstance(DATEFORMAT, System
    .Utils.gmtTimeZone, Locale.US);
  private Timer timer = null;
  private Object lock = new Object();
  // WebSocket listener
  class MyWebSocketListener extends WebSocketEventNotifyBase {
    @Override
    public void onCreate(IWebSocketSession webSocketSession) {
      WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME + "#MyWebSocketListener.onCreate" + webSocketSession.getSessionId());
    }
    @Override
    public void onDestroy(IWebSocketSession webSocketSession) {
    }
  }
}
```
@Override
public void onMessage(IWebSocketSession webSocketSession, WebSocketMessage message) {
    // echo messages we receive back to the browser
    if (message.isText()) {
        WebSocketMessage messageText = WebSocketMessage.createMessageText(webSocketSession.isMaskOutgoingMessages(), message.getValueString());
        webSocketSession.sendMessage(messageText);
        WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME + "#MyWebSocketListener.onMessage[" + webSocketSession.getSessionId() + "][text]: " + message.getValueString());
    } else if (message.isBinary()) {
        WebSocketMessage messageBinary = WebSocketMessage.createMessageBinary(webSocketSession.isMaskOutgoingMessages(), message.getBuffer(), message.getOffset(), message.getLen());
        webSocketSession.sendMessage(messageBinary);
        WMSLoggerFactory.getLogger(CLASS).info(CLASSNAME + "#MyWebSocketListener.onMessage[" + webSocketSession.getSessionId() + "][binary]: #" + BufferUtils.encodeHexString(message.getBuffer(), message.getOffset(), message.getLen()));
    }
}

class MyTimerTask extends TimerTask {
    private IVHost vhost = null;
    private WebSocketContext webSocketContext = null;

    MyTimerTask(IVHost vhost) {
        this.vhost = vhost;
        this.webSocketContext = vhost.getWebSocketContext();
    }

    @Override
    public void run() {
        String messageStr = Server.getInstance().getVersion() + " date:" + fastDateFormat.format(new Date()) + " GMT";
        // broadcast message to all active sessions attached to this HTTPProvider
        WebSocketMessage messageText = WebSocketMessage.createMessageText(webSocketContext.isMaskOutgoingMessages(), messageStr);
        broadcastWebSocketMessage(messageText);
    }
}
public void onHTTPRequest(IVHost vhost, IHTTPRequest req, IHTTPResponse resp) {
    if (!doHTTPAuthentication(vhost, req, resp))
        return;
    synchronized(lock) {
        // create timer task on first connection
        if (this.timer == null) {
            this.timer = new Timer();
            timer.scheduleAtFixedRate(new MyTimerTask(vhost), TIMER_INTERVAL, TIMER_INTERVAL);
        }
    }
    // is this an upgrade request
    if (req.isUpgradeRequest()) {
        // it this an websocket upgrade request
        String upgradeType = req.getHeader("upgrade");
        if (upgradeType != null && upgradeType.equalsIgnoreCase(IWebSocketSession.HTTPHEADER_NAME)) {
            // set response header to accept the upgrade
            resp.setHeader("Upgrade", IWebSocketSession.HTTPHEADER_NAME);
            resp.setHeader("Connection", "Upgrade");
            // set the security hash
            String webSocketKey = req.getHeader(IWebSocketSession.HTTPHEADER_SECKEY);
            if (webSocketKey != null) {
                String digestStr = WebSocketUtils.createSecResponse(webSocketKey);
                if (digestStr != null)
                    resp.setHeader(IWebSocketSession.HTTPHEADER_SECACCEPT, digestStr);
            }
            // set 101 response code to accept upgrade request
            resp.setResponseCode(101);
            // insert WebSocket listener for this session
            resp.setUpgradeRequestProtocol(IHTTPResponse.UPGRADE_PROTOCOL_WEBSOCKETS, new MyWebSocketListener());
        }
    } else {
        resp.setResponseCode(404); // return 404 if not websocket upgrade request
    }
}
This HTTP provider echos back the WebSocket messages it receives and periodically broadcasts a text WebSocket message to all connected sessions.

**Send and receive WebSocket messages**

WebSocket messages are received by a listener class that implements the `IWebSocketEventNotify` interface. We recommend that you create a class that extends the `WebSocketEventNotifyBase` and overrides the callback methods you want to use.

There are two types of WebSocket messages: binary and text.

- A text WebSocket message is created using the following API:

  ```java
  WebSocketMessage.createMessageText(boolean mask, String messageStr);
  ```

- A binary WebSocket message is created using one of the following APIs:

  ```java
  WebSocketMessage.createMessageBinary(boolean mask, byte[] buffer);
  WebSocketMessage.createMessageBinary(boolean mask, byte[] buffer, int offset, int len);
  ```

A WebSocket message can be sent over an individual WebSocketSession using the following API:

```
webSocketSession.sendMessage(WebSocketMessage message);
```

An HTTP provider maintains a list of active WebSocket sessions. A message can be broadcast to all sessions of a given HTTP provider using the following API:

```
httpProvider.broadcastWebSocketMessage(WebSocketMessage message);
```

**View WebSocket messages in a browser**

With an HTTP provider, you can view information from your own specified URL. For example:
The following is an example of HTML and JavaScript that corresponds to the WebSocket HTTP provider above, and can be used to send and receive message to a web browser. This file must be hosted on a web server as an HTML file. You need to change the `wsURL` variable to point to your Wowza Streaming Engine instance that’s running the WebSocket HTTP provider:

```html
Connect
Send Message
Hello, Wowza Streaming Engine!
```

More resources

- Make an HTML page with graphics and scripts using an HTTP provider in the Wowza Streaming Engine Java API