WebSocket in PNaCl application

Published 2014-10-28 | (Compatible with SDK 4.5,5.0,5.1 and 2013,2014 models)

> This documents provides information about using WebSocket in SmartTV PNaCl application

Contents

- Prerequisites
- Introduction
- Interface description
  - The pp::WebSocket interface
- Using tutorial applications
  - Run server
  - Run client
- Create your own PNaCl client

This document describes how to use WebSockets in a SmartTV PNaCl application. Tutorial shows:

WebSocket Client - PNaCl sample application base on pp::WebSocket interface.
WebSocket Server - sample WebSocket server written in Node.js.

Prerequisites

To run client application described in this tutorial, please download the **client application source code** and extract it into the Smart TV SDK Emulator application folder. Start server application requires to download **Node.js installer** for your platform and install. Afterwards please download the **server application source code** and extract it at any location on the disk.

To create a PNaCl application NaCl SDK and text editor are needed. More information about creating simple PNaCl application can be found in How to create sample PNaCl application.

Introduction

Client application presents a way of creating WebSocket connecting with WebSocket Server, which encodes given message with Caesar cipher. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions the alphabet. In this case shift is equal to 3. Client application allows to send text and receive feedback with server.

Server is not a PNaCl application. It uses Node.js definitions of http server and WebSocket server. Logging is achieved through JavaScript built-in console object, which writes to standard output. Node.js is software platform provide way to build network applications both server-side and client-side.

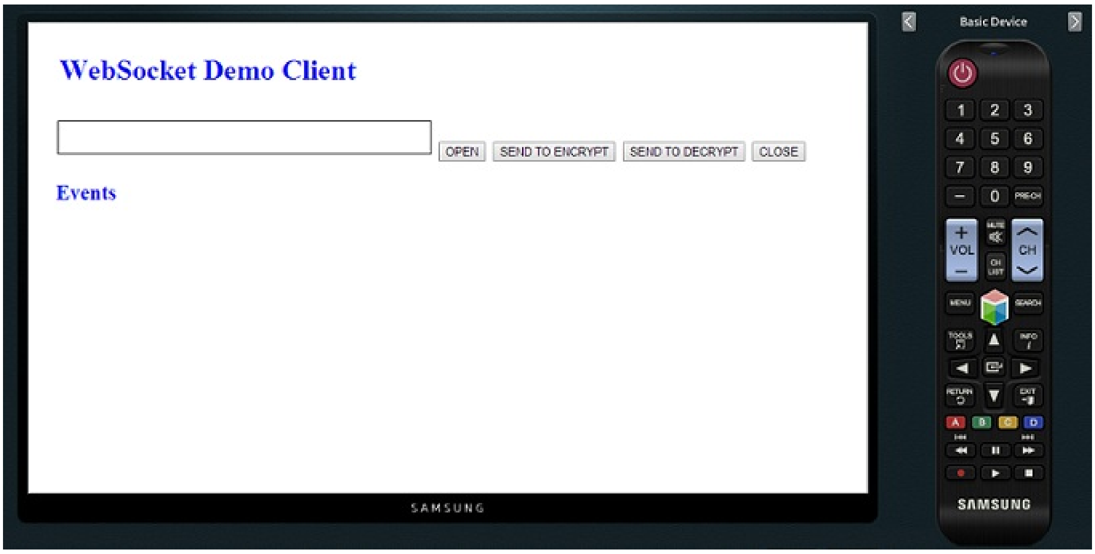
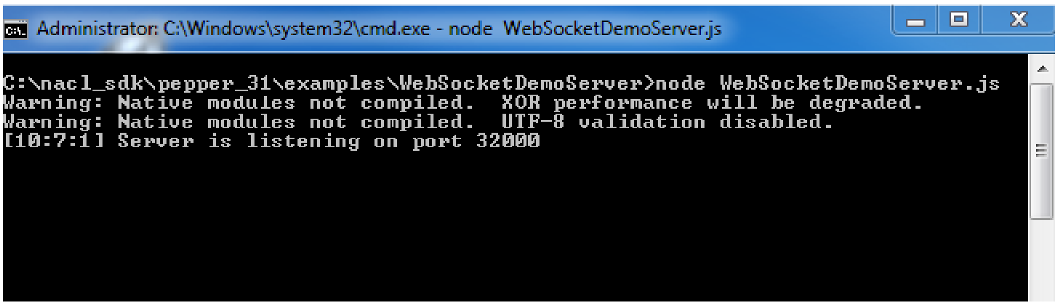


Figure 1: Client application view after initialization.


Administrator: C:\Windows\system32\cmd.exe - node WebSocketDemoServer.js

```
C:\nacl_sdk\pepper_31\examples\WebSocketDemoServer>node WebSocketDemoServer.js
Warning: Native modules not compiled.  XOR performance will be degraded.
Warning: Native modules not compiled.  UTF-8 validation disabled.
[10:7:11] Server is listening on port 32000
```

Figure 2: Server application view after initialization.

Interface description

WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application.

The pp::WebSocket interface

The pp::WebSocket interface in PNaCl providing bi-directional, full-duplex, communications over a single TCP socket.

Using tutorial applications

Run server

After installation of Node.js platform to start the WebSocket Server is necessary to:

1. Make sure the installation folder is added to the PATH environment variable.
2. Go to the WebSocket Server project folder.
3. Install websocket package using command: > npm install websocket -g.
4. Link websocket package into project using command: > npm link websocket.
5. Run WebSocket Server using command: > node WebSocketDemoServer.js.

> Note
>
> Server start working default on localhost port 32000.

Run client

When application loads, user options will be:

Open

Used to open a connection between WebSocket Client and WebSocket Server. If WebSocket Server is not running on the Events section will show appropriate message.

Send to encrypt (decrypt)

Used to send message entered in TextArea to the server. This message will be encrypt(decrypt) by using Caesar cipher and send back.

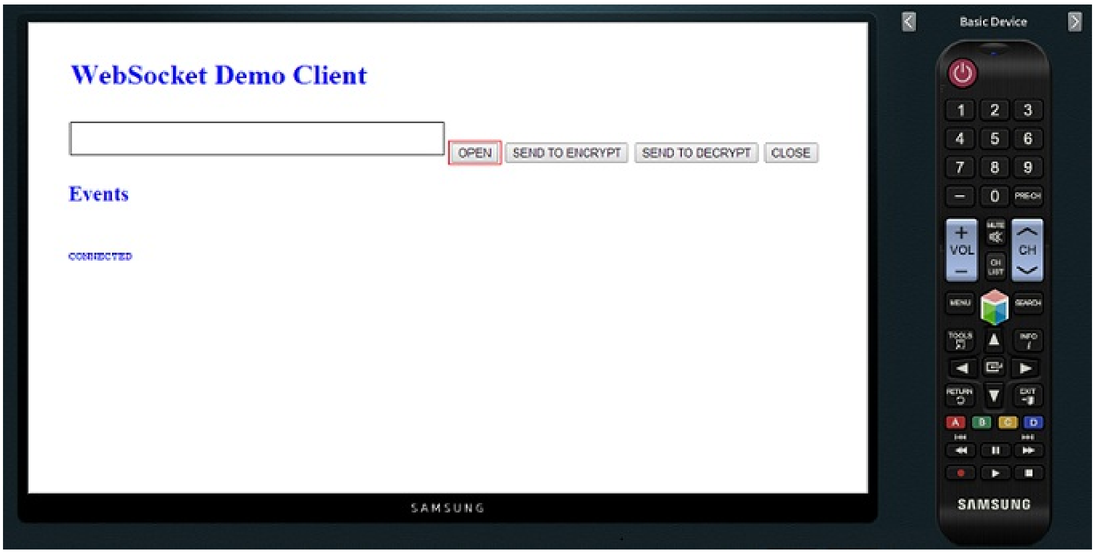
Close

Used to close connection.

Example of use client application and responses of server are shown in figures below:

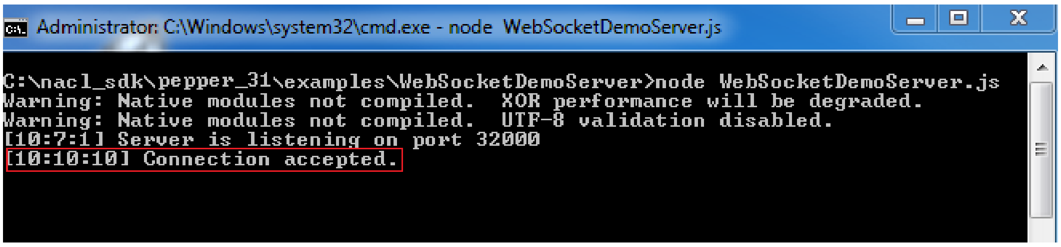
1. Open connection

   To open connection use Open button. If WebSocket Server is running, the connection will be successful and in Events section appears “CONNECTED” statement. Can also be observed appropriate statement in server console.



CONNECTED

Figure 3: Opening connection between WebSocket Server and WebSocket Client.

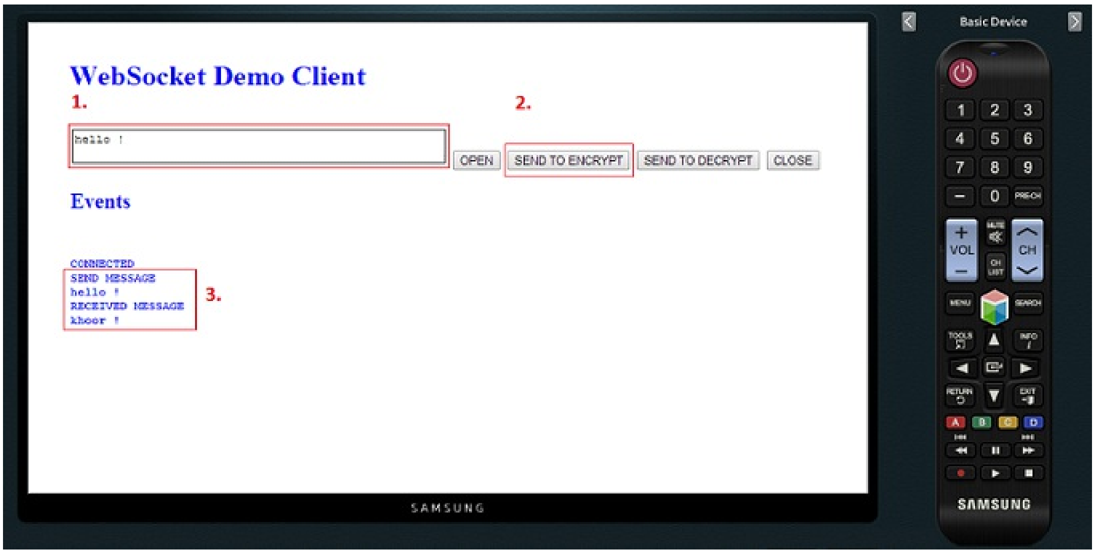

Administrator: C:\Windows\system32\cmd.exe - node WebSocketDemoServer.js

```
C:\nacl_sdk\pepper_31\examples\WebSocketDemoServer>node WebSocketDemoServer.js
Warning: Native modules not compiled.  XOR performance will be degraded.
Warning: Native modules not compiled.  UTF-8 validation disabled.
[10:7:1] Server is listening on port 32000
[10:10:10] Connection accepted.
```

Figure 4: Server response to establish connection.

1. Send message to encrypt

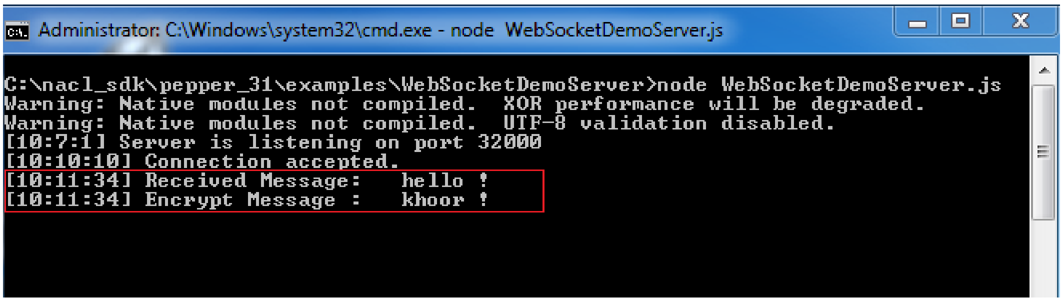
   Afterwards fill TextArea and click Send to encrypt button. There also will be shown the query result.



hello !

CONNECTED
SEND MESSAGE
hello !
RECEIVED MESSAGE
khoor !

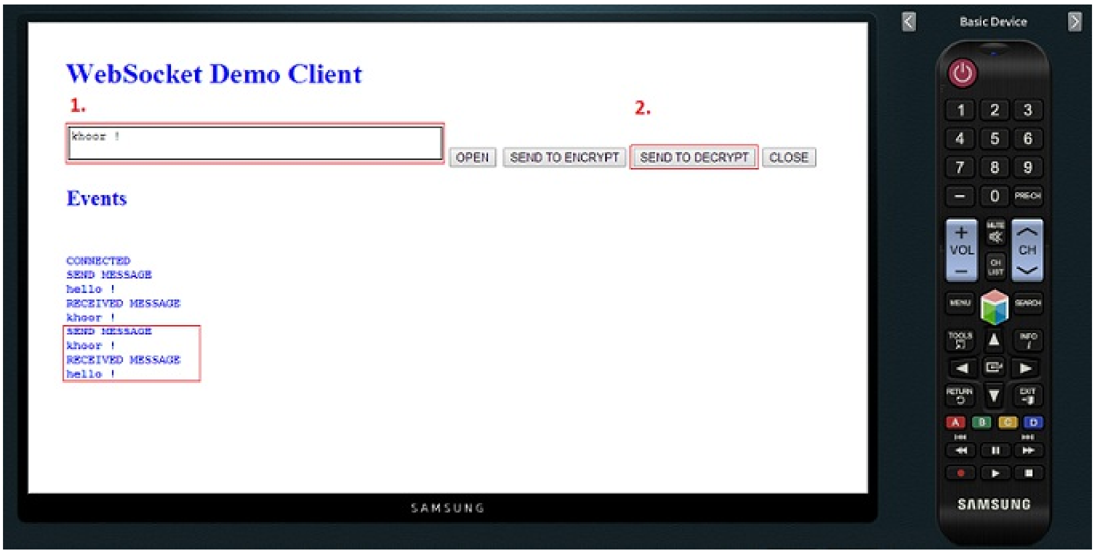
Figure 5: Sending message to encrypt.


Administrator: C:\Windows\system32\cmd.exe - node WebSocketDemoServer.js

```
C:\nacl_sdk\pepper_31\examples\WebSocketDemoServer>node WebSocketDemoServer.js
Warning: Native modules not compiled.  XOR performance will be degraded.
Warning: Native modules not compiled.  UTF-8 validation disabled.
[10:7:1] Server is listening on port 32000
[10:10:10] Connection accepted.
[10:11:34] Received Message:    hello !
[10:11:34] Encrypt Message :    khoor !
```

Figure 6: Server response.

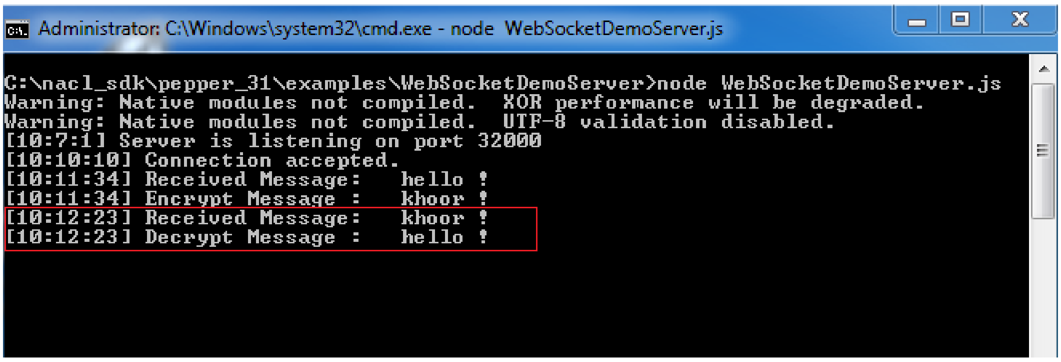
1. Send message to decrypt



```
khoor !
```

```
CONNECTED
SEND MESSAGE
hello !
RECEIVED MESSAGE
khoor !
SEND MESSAGE
khoor !
RECEIVED MESSAGE
hello !
```

Figure 7: Sending message to decrypt.

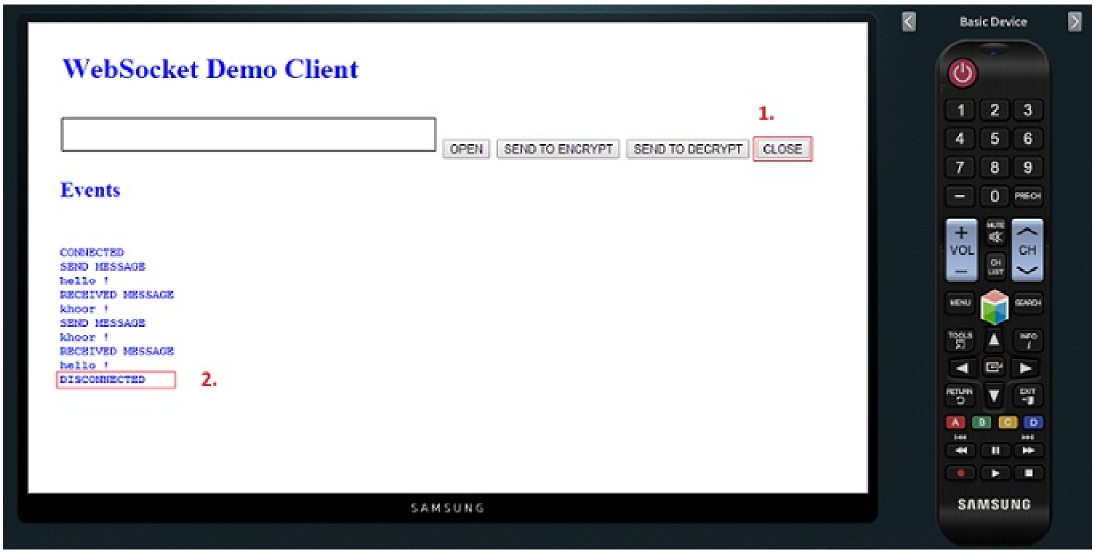

Administrator: C:\Windows\system32\cmd.exe - node WebSocketDemoServer.js

```
C:\nacl_sdk\pepper_31\examples\WebSocketDemoServer>node WebSocketDemoServer.js
Warning: Native modules not compiled.  XOR performance will be degraded.
Warning: Native modules not compiled.  UTF-8 validation disabled.
[10:7:1] Server is listening on port 32000
[10:10:10] Connection accepted.
[10:11:34] Received Message:    hello !
[10:11:34] Encrypt Message :    khoor !
[10:12:23] Received Message:    khoor !
[10:12:23] Decrypt Message :    hello !
```

Figure 8: Server response.

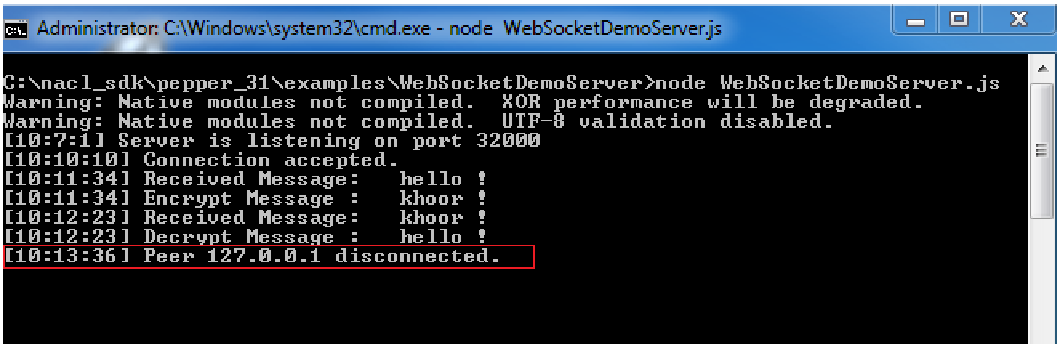
1. Close connection

Finally close connection using Close button. Server will record it.



```
CONNECTED
SEND MESSAGE
hello !
RECEIVED MESSAGE
khoor !
SEND MESSAGE
khoor !
RECEIVED MESSAGE
hello !
DISCONNECTED
```

Figure 9: Closing connection.


Administrator: C:\Windows\system32\cmd.exe - node WebSocketDemoServer.js

```
C:\nacl_sdk\pepper_31\examples\WebSocketDemoServer>node WebSocketDemoServer.js
Warning: Native modules not compiled.  XOR performance will be degraded.
Warning: Native modules not compiled.  UTF-8 validation disabled.
[10:7:1] Server is listening on port 32000
[10:10:10] Connection accepted.
[10:11:34] Received Message:   hello !
[10:11:34] Encrypt Message :   khoor !
[10:12:23] Received Message:   khoor !
[10:12:23] Decrypt Message :   hello !
[10:13:36] Peer 127.0.0.1 disconnected.
```

Figure 10: Server response to close connection.

Create your own PNaCl client

In order to be able to create a client, make sure that the header file containing pp::WebSocket interface is included in your client source file.

To connect to the server, it is necessary to create a pointer to pp::WebSocket object using pp::Instance and call a Connect() method. You have to pass four parameters to this function:

A Var of string type representing a WebSocket server URL

A pointer to an array of Var of string type specifying sub-protocols. Each Var represents one sub-protocol. In this case WebSocket Server accept only “Caesar-protocol”.

Number of sub-protocols in protocols

A CompletionCallback

```
pp::Var protocols[] =
     { pp::Var("caesar-protocol") };
pp::CompletionCallback callback = factory.NewCallback(&WebSocketDemoClientInstance::OpenCompletion);
pp::WebSocket* websocket = new pp::WebSocket(this);
websocket->Connect(pp::Var("ws://localhost:32000/"), protocols, 1, callback);
```

Now is possible to send and receive message from the server. For this purpose use SendMessage() and ReceiveMessage() functions.

```
std::string message = "hello";
websocket->SendMessage(pp::Var(message));
pp::Var receivedMessage;
pp::CompletionCallback callback = factory.NewCallback(&WebSocketDemoClientInstance::ReceiveCompletion);
websocket->ReceiveMessage(&receivedMessage, callback);
```

To receive the message you need create a callback. The callback function must accept a parameter of type int32_t. Negative result indicates an error.

> Important

> If error codes are returned from socket functions, the meanings of all PPAPI error codes can be found in pp_errors.h file.

Close() closes the specified WebSocket connection by specifying code and reason. For the usual case must be used PP_WEBSOCKETSTATUSCODE_NORMAL_CLOSURE.

```
    pp::CompletionCallback callback = factory.NewCallback(&WebSocketDemoClientInstance::CloseCompletion);
    websocket->Close(PP_WEBSOCKETSTATUSCODE_NORMAL_CLOSURE, pp::Var("close"), callback);
```