TCP Socket 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 TCP socket in SmartTV NaCl application

Contents

This document describes how to use TCP sockets in a SmartTV application. Tutorial shows two sample PNaCl applications - the client and the server. The pp::TCPSocketPrivate and pp::TCPServerSocketPrivate interfaces were used to create those applications. Both can be tested on Smart TV with Native Client, Smart TV SDK Emulator or Google Chrome browser.

Prerequisites

To run applications described in this tutorial, please download the **application source code** and extract it into the Smart TV SDK Emulator application folder. If the Server and Client application are going to be tested on the same computer, applications should be run on separate virtual machines.

To create a PNaCl application NaCl SDK and text editor are needed. Make sure that files contain pp::TCPSocketPrivate and pp::TCPServerSocketPrivate interfaces are in included directory. More information about creating simple PNaCl application can be found in How to create sample PNaCl application.

Introduction

Applications present a simple chat allowing communication between all users based on TCP sockets. Clients send messages to the server and the server forwards it to all registered users. Additionally the list of all chat participants is displayed in the client application, so that each user can see who else is available. There is no limited number of users of the application.

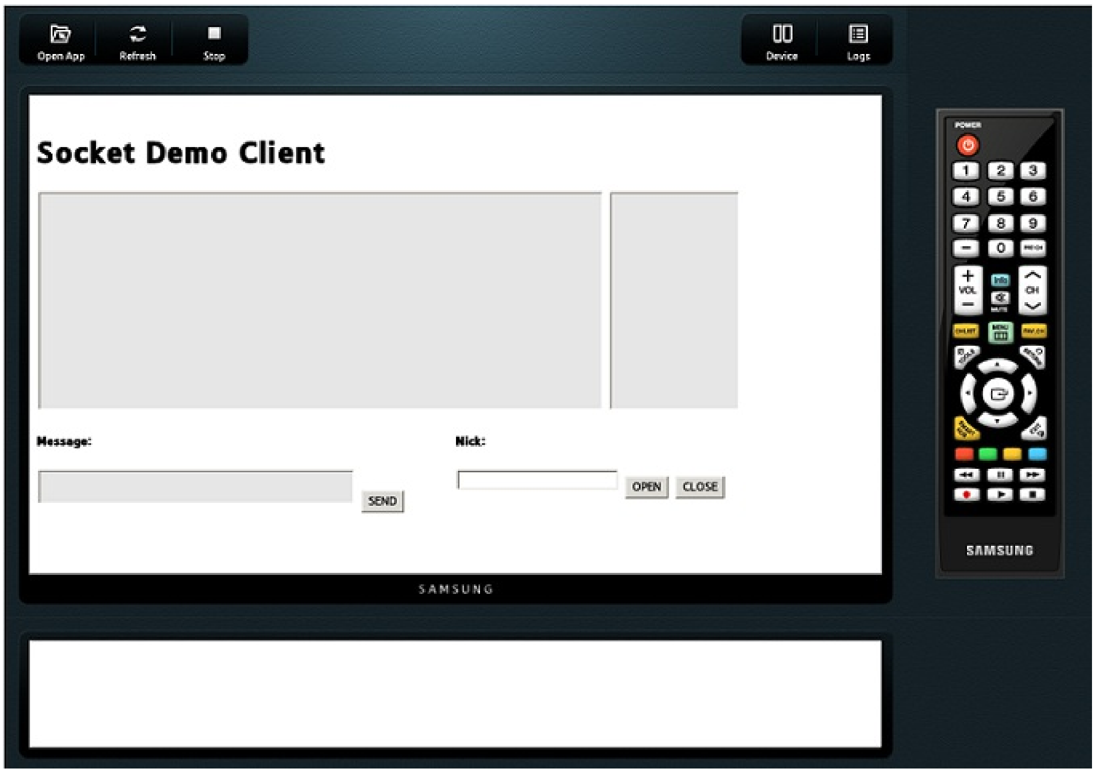
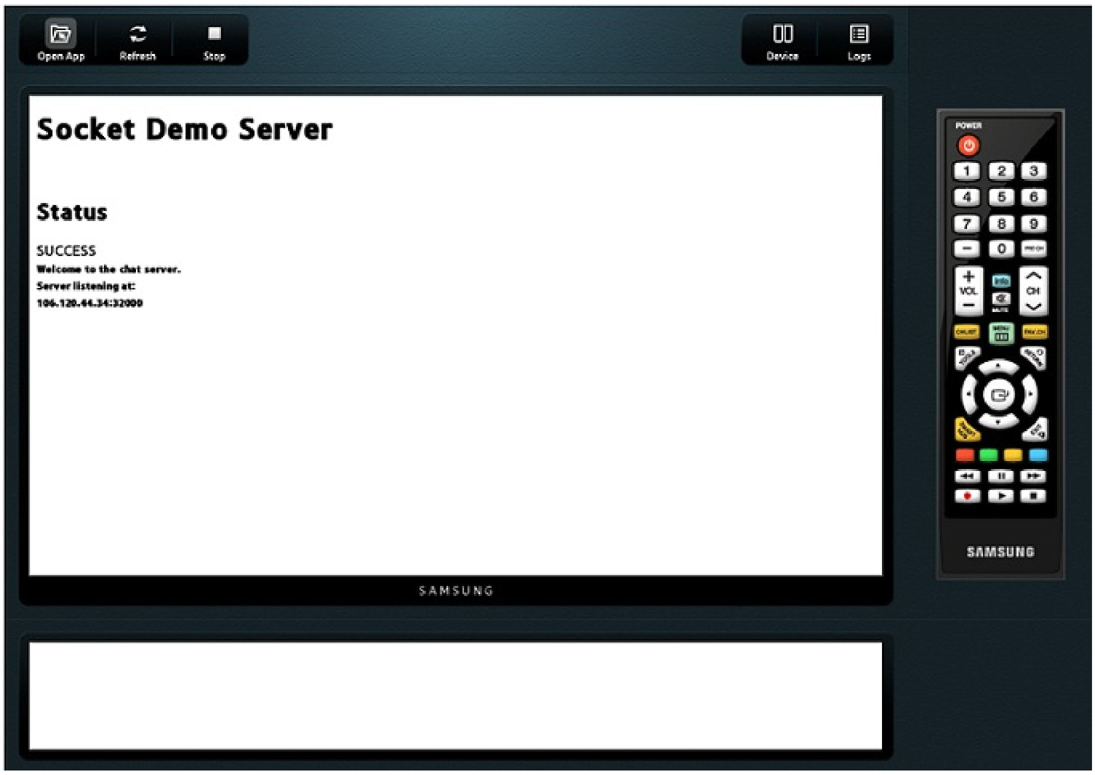


Figure 1: Client application view after initialization



SUCCESS
Welcome to the chat server.
Server listening at:
106.120.44.34:32000

Figure 2: Server application view after initialization

Interfaces description

pp::TCPSocketPrivate and pp::TCPServerSocketPrivate interfaces used in application are private. That is mean need more tests and may change without notice.

> Important
>
> Before starting application development it is necessary to check that interfaces are present in an included directory.

The pp::TCPSocketPrivate interface

Interface provides TCP socket operations, allowing to control and use connection-oriented sockets, which use Transmission

Control Protocol.

The pp::TCPServerSocketPrivate interface

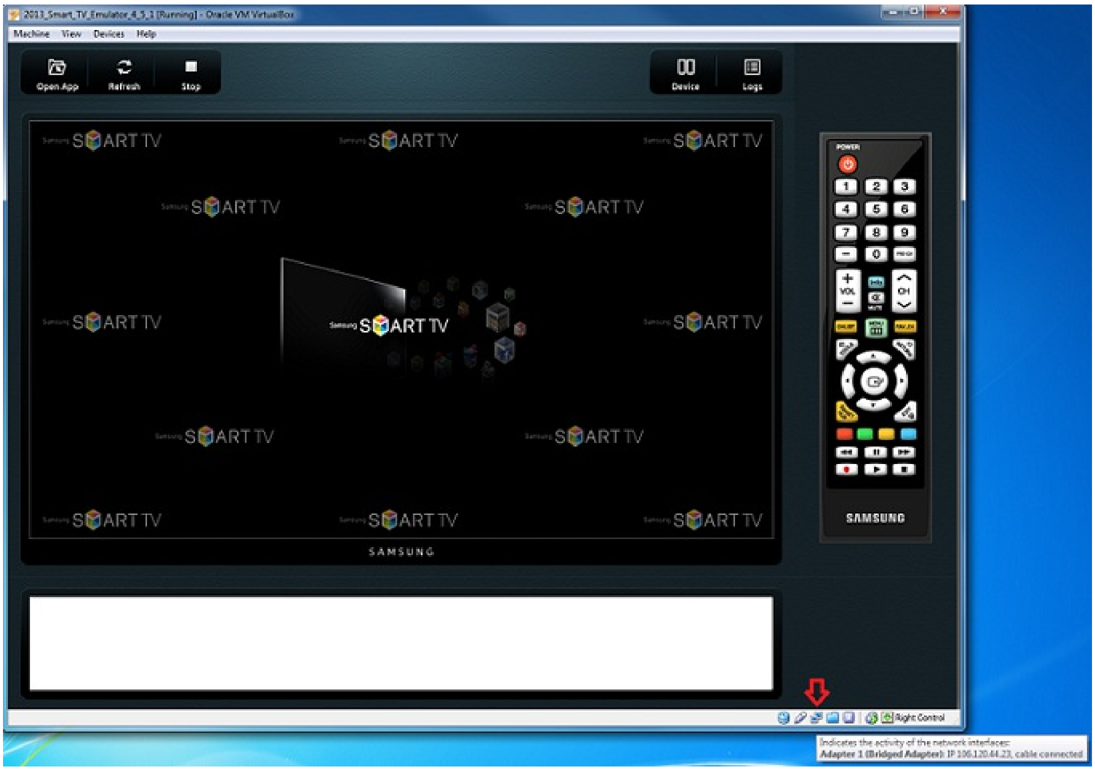
Interface provides TCP socket operations and features specific to the server such as Listen() and Accept().

Using tutorial applications

Run Server

If you are using VirtualBox make sure you change network settings from NAT to Bridged Adapter. Otherwise your server will not be able to receive messages from clients. More information about setting Bridged Network can be found in Using the Smart TV Emulator with VirtualBox & Troubleshooting.

After that you will have to check your server's IP address. You are going to use this address in both server and client applications. In server you will set up the server socket with the address, and in client you will connect to this address. Once you check virtual machine's IP address, you will have to update it in the index.html file (host attribute of the <embed> element) both on the client side and the server side.



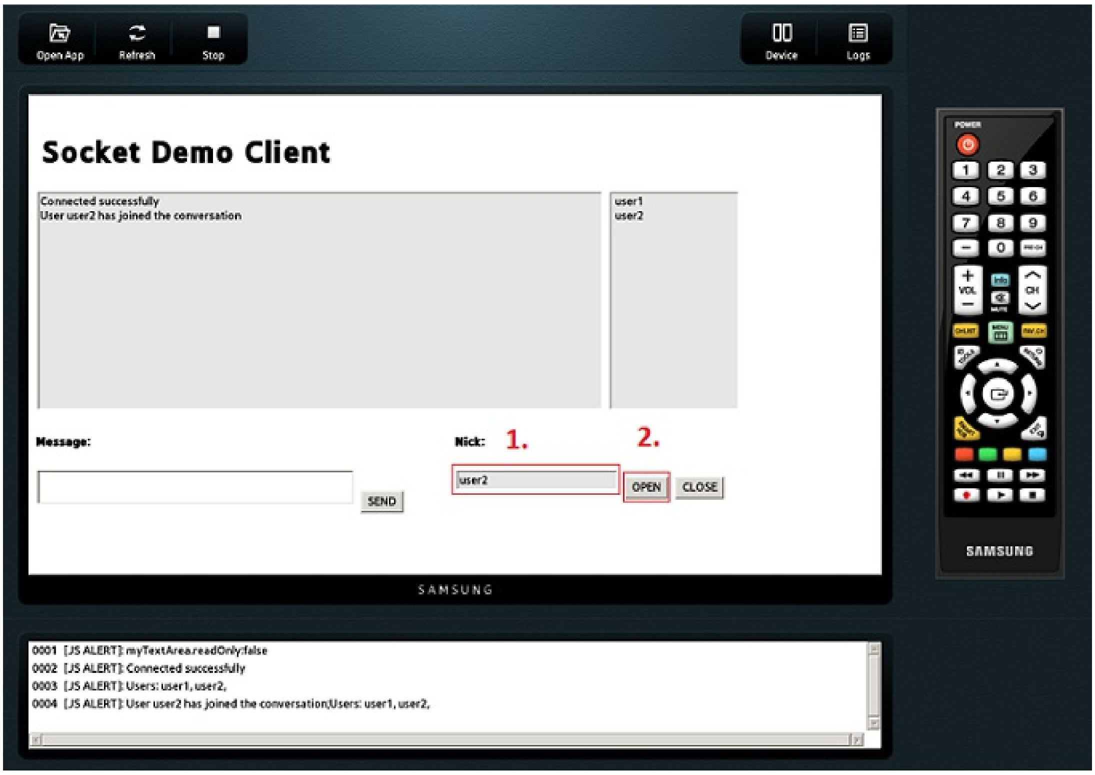
Indicates the activity of the network interfaces:
Adapter 1 (Bridged Adapter): IP 106.120.44.23, cable connected

Figure 3: Checking virtual machine IP address

Run Client

Client is a PNaCl application which should be run on a separate virtual machine with Smart TV emulator. When application loads, user options will be:
Open button
Used to open a connection between client and server. Before clicking this button you must fill the user nickname in "Nick" TextArea. If user nickname is already taken, you will receive an appropriate message.
Send button
Used to send message entered in "Message" TextArea to the server. For this purpose you can use enter button too.
Close button
Used to close the current connection.
Main area of text
Place where received messages are displayed.
Area of nicknames
Place where users nicknames are displayed.
Example of use Client application is shown in figures below:

1. Fill the user nickname and click open button. Nickname must be unique. If server is running, the connection process will be successful.



Socket Demo Client

```
Connected successfully
User user2 has joined the conversation
```

```
user1
user2
```

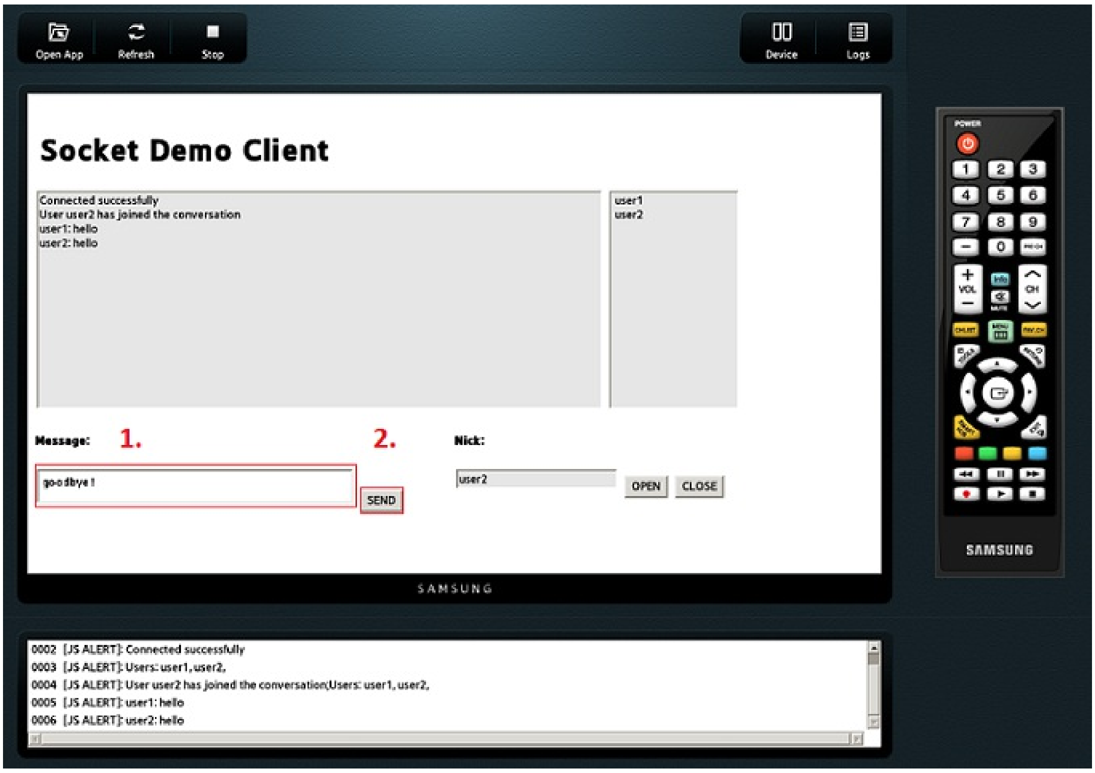
Message:

Nick: user2

```
0001  [JS ALERT]: myTextArea.readOnly:false
0002  [JS ALERT]: Connected successfully
0003  [JS ALERT]: Users: user1, user2,
0004  [JS ALERT]: User user2 has joined the conversation,Users: user1, user2,
```

Figure 4: Opening connection

1. Send message



Socket Demo Client

```
Connected successfully
User user2 has joined the conversation
user1: hello
user2: hello
```

```
user1
user2
```

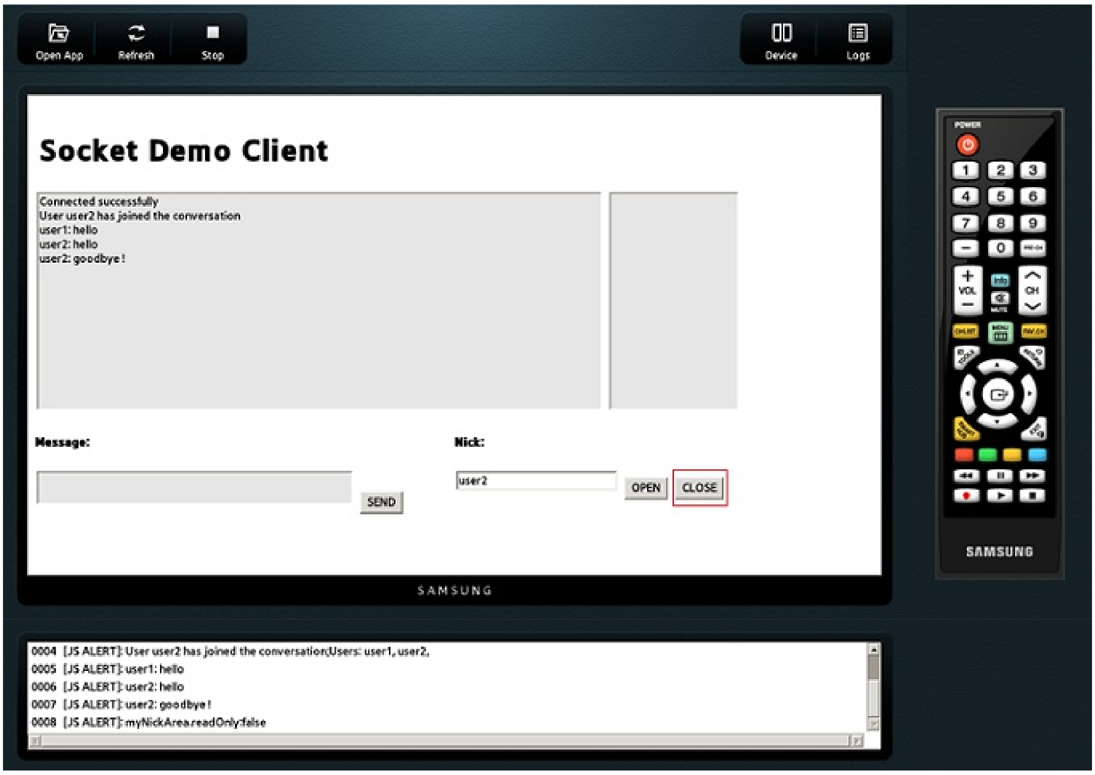
Message: goodbye !

Nick: user2

```
0002  [JS ALERT]: Connected successfully
0003  [JS ALERT]: Users: user1, user2,
0004  [JS ALERT]: User user2 has joined the conversation,Users: user1, user2,
0005  [JS ALERT]: user1: hello
0006  [JS ALERT]: user2: hello
```

Figure 5: Sending message

1. After conversation close connection



```
Connected successfully
User user2 has joined the conversation
user1: hello
user2: hello
user2: goodbye !
```

Nick: user2

```
0004  [JS ALERT]: User user2 has joined the conversation;Users: user1, user2,
0005  [JS ALERT]: user1: hello
0006  [JS ALERT]: user2: hello
0007  [JS ALERT]: user2: goodbye !
0008  [JS ALERT]: myNickArea.readOnly:false
```

Figure 6: Client view after close connection

Create your own application

Create your own PNaCl Server

In order to be able to create a server, make sure that header files containing required interfaces are present in an included directory.

The headers are:

tcp_server_socket_private.h
tcp_socket_private.h
net_address_private.h

Once you have those files, include them in your server source file. In the application, you need to create a pp::TCPServerSocketPrivate object, passing an instance handle to the constructor. After that you need to call the Listen() method. It requires three arguments:

PP_NetAddress_Private structure containing server's IP address and port
backlog defining maximum number of connections processed
callback function to be called after completing Listen() method

The first parameter needs to be created with a CreateFromIPv4Address function. You have to pass three parameters to this function:

integer array with IP address parts
port number
empty PP_NetAddress_Private structure

Calling this method will save the IP address and port in the given structure, passed by pointer.

```
PP_NetAddress_Private serverAddress;
bool addressCreated = pp::NetAddressPrivate::CreateFromIPv4Address(ipAddress, portNumber, &serverAddress);
```

After you create the first parameter for Listen() method, you define a backlog (i.e. 100) and the callback function. This function must accept a parameter of type int32_t. This will be the result of the calling method in this case Listen(). Negative result indicates an error.

```
int32_t res = serverSocket->Listen(&serverAddress, backlog, factory.NewCallback(&ChatServer::OnListenCompleted)
);
```

If result equals 0, Accept() method should be called with a similar callback. Accept() needs an empty PP_Resource for storing socket data and a callback.

```
PP_Resource socket;
int32_t res = serverSocket->Accept(&socket, factory.NewCallback(&ChatServer::OnAcceptCompleted, &socket));
```

As you can see, the PP_Resource is passed to the callback function. After that Read() method needs to be called to get data from the connected client. Read() method needs a buffer to keep data, maximum message size and again a callback function. Please note that while in Listen() and Accept() methods, 0 is the value indicating successful completion, Read() method returns a number of characters read, so in this case it should return a positive value.

```
char* messageBuffer = new char[maxMessageSize];
int32_t res = newUser->socket->Read(messageBuffer, maxMessageSize, factory.NewCallback(&ChatServer::OnReadCompleted, newUser, false));
```

The callback function gets data of the new user and a flag indicating whether Accept() needs to be called again (you can implement it differently). At this point you can process the received message in any way you want to.

Create your own PNaCl Client

In order to be able to create a Client, make sure that the header file containing required interface is present in an included directory. In the case of the client the file is: tcp_socket_private.h. Once you have this file, include it in your client source file.

If you want to connect to the Server, it is necessary to create a pp::TCPSocketPrivate object, passing an instance handle to the constructor. An instance handle identifies an instance in a constructor for a resource. instanceHandle occurring in the code below is a pp::InstanceHandle object.

After that it is possible to call the Connect() method. Parameters host and port indicate server IP address and listening port. The callback function must accept a parameter of type int32_t. This will be the result of the calling Connect() method. Negative result indicates an error.

```
void ChatClient::Connect()
{
  pp::TCPSocketPrivate socket = pp::TCPSocketPrivate(instanceHandle);
  pp::CompletionCallback callback = factory.NewCallback(&ChatClient::OpenCompletion);
  socket.Connect(host, port, callback);
}
```

In the next step you can send and receive messages from the server. For this purpose use Write() and Read() functions. In both cases you must create a callback.

```
void ChatClient::Write(const std::string& message)
{
  pp::CompletionCallback callback = factory.NewCallback(&ChatClient::WriteCompletion);
  std::string sentMessage = message;
  int32_t bytes_to_write = sentMessage.size();
  socket.Write(sentMessage.c_str(), bytes_to_write, callback);
}
void ChatClient::Read()
{
  pp::CompletionCallback callback = factory.NewCallback(&ChatClient::ReadCompletion);
  socket.Read(buffer, sizeof(buffer), callback);
}
```

factory occurring in the code is a pp::CompletionCallbackFactory<ChatClient> object.

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