Lab 6
RPC Programming

Guideline

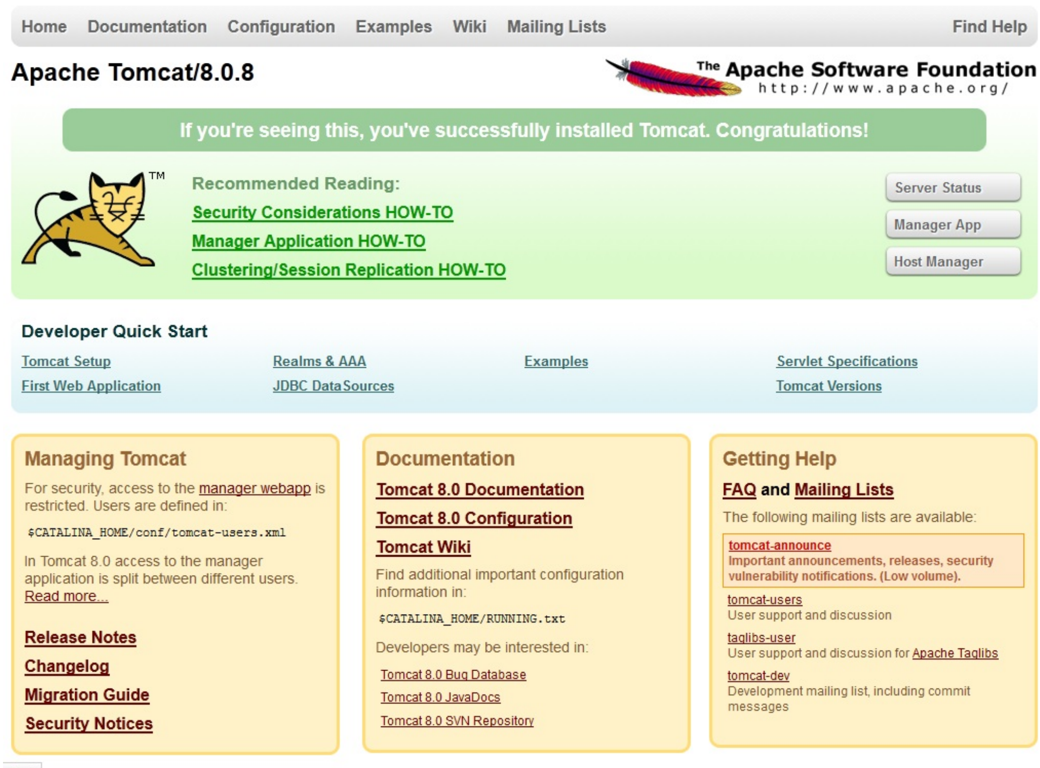
- Read text materials and practice TCP/UDP Socket Programming
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- **Objectives**
  - Understand the UDP server/client programming mechanism
  - Learn the usage of multi-process, multi-thread and I/O multiplexing
- **Practice**

1. UDP Programming Test
2. Multi-process server programming
3. I/O multiplexing (Select / Poll) server programming

Running the Application Example

Apache Tomcat

- Apache Tomcat, often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF).
- Tomcat implements several Java EE specifications including Java Servlet and JavaServer Pages (JSP), and provides a HTTP web server environment

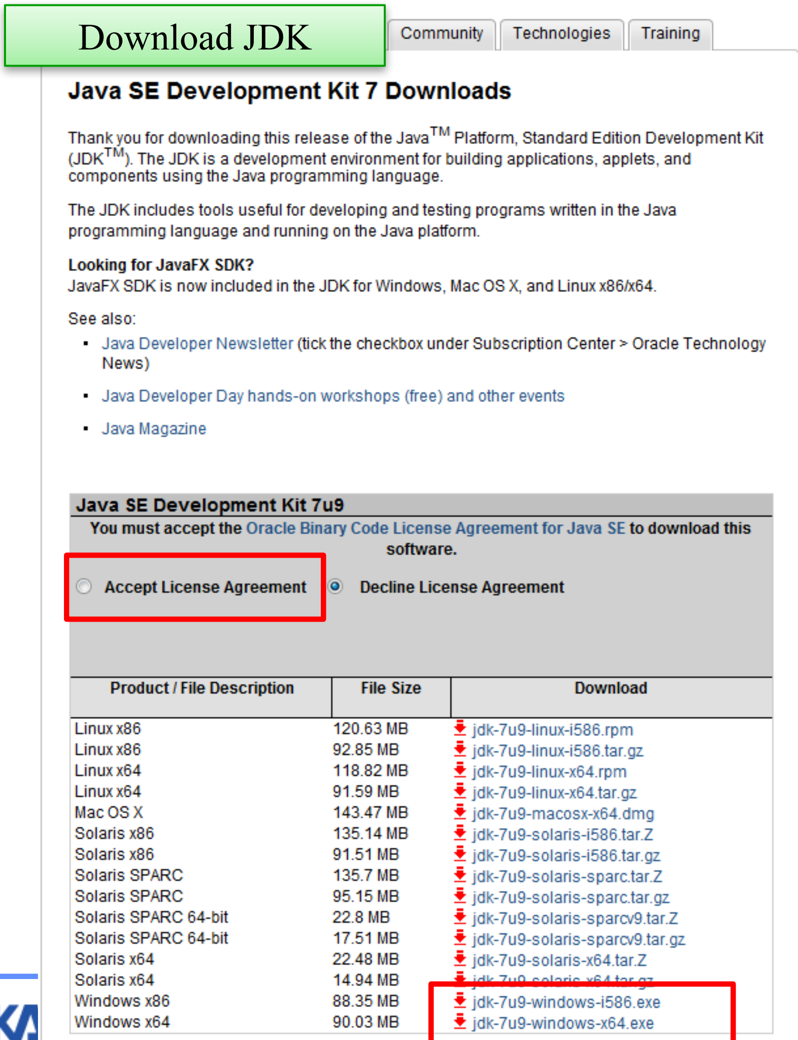


Setup Steps for Tomcat development environment

- Download & Install Java S/W Development Kit
- Download a server (Apache Tomcat 7)
- Test server (Apache Tomcat 7)
- Set up development environment
- Create custom Web Application
- Deploy a Web Application

Download & Install JDK 7

- http://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html



Download JDK

Community | Technologies | Training

Java SE Development Kit 7 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Looking for JavaFX SDK?
JavaFX SDK is now included in the JDK for Windows, Mac OS X, and Linux x86/x64.

See also:

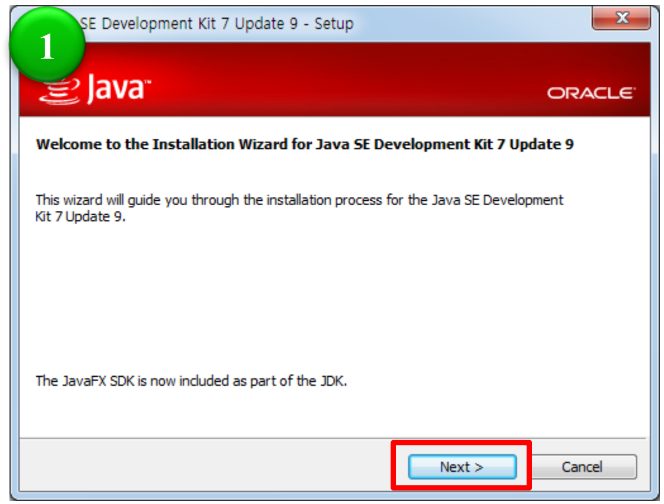
- Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

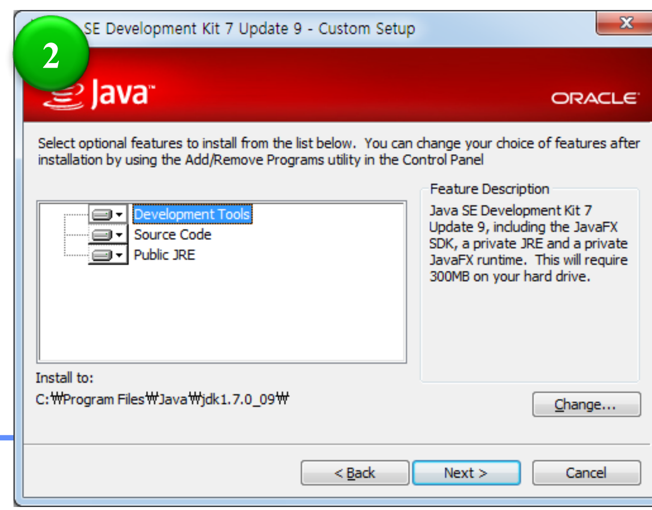
Java SE Development Kit 7u9

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

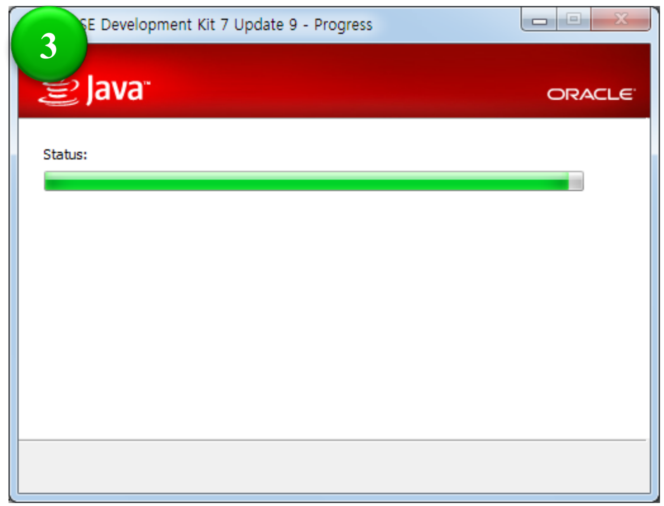
○ **Accept License Agreement** ◉ **Decline License Agreement**

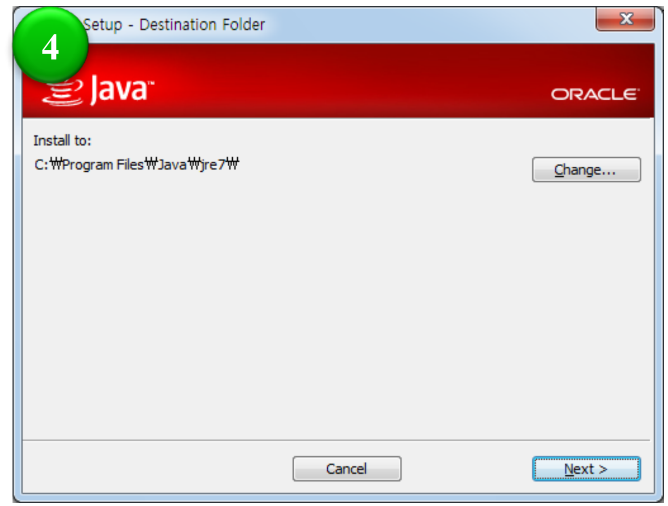
Product / File Description	File Size	Download
Linux x86	120.63 MB	jdk-7u9-linux-i586.rpm
Linux x86	92.85 MB	jdk-7u9-linux-i586.tar.gz
Linux x64	118.82 MB	jdk-7u9-linux-x64.rpm
Linux x64	91.59 MB	jdk-7u9-linux-x64.tar.gz
Mac OS X	143.47 MB	jdk-7u9-macosx-x64.dmg
Solaris x86	135.14 MB	jdk-7u9-solaris-i586.tar.Z
Solaris x86	91.51 MB	jdk-7u9-solaris-i586.tar.gz
Solaris SPARC	135.7 MB	jdk-7u9-solaris-sparc.tar.Z
Solaris SPARC	95.15 MB	jdk-7u9-solaris-sparc.tar.gz
Solaris SPARC 64-bit	22.8 MB	jdk-7u9-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.51 MB	jdk-7u9-solaris-sparcv9.tar.gz
Solaris x64	22.48 MB	jdk-7u9-solaris-x64.tar.Z
Solaris x64	14.94 MB	jdk-7u9-solaris-x64.tar.gz
Windows x86	88.35 MB	jdk-7u9-windows-i586.exe
Windows x64	90.03 MB	jdk-7u9-windows-x64.exe





Install JDK 7





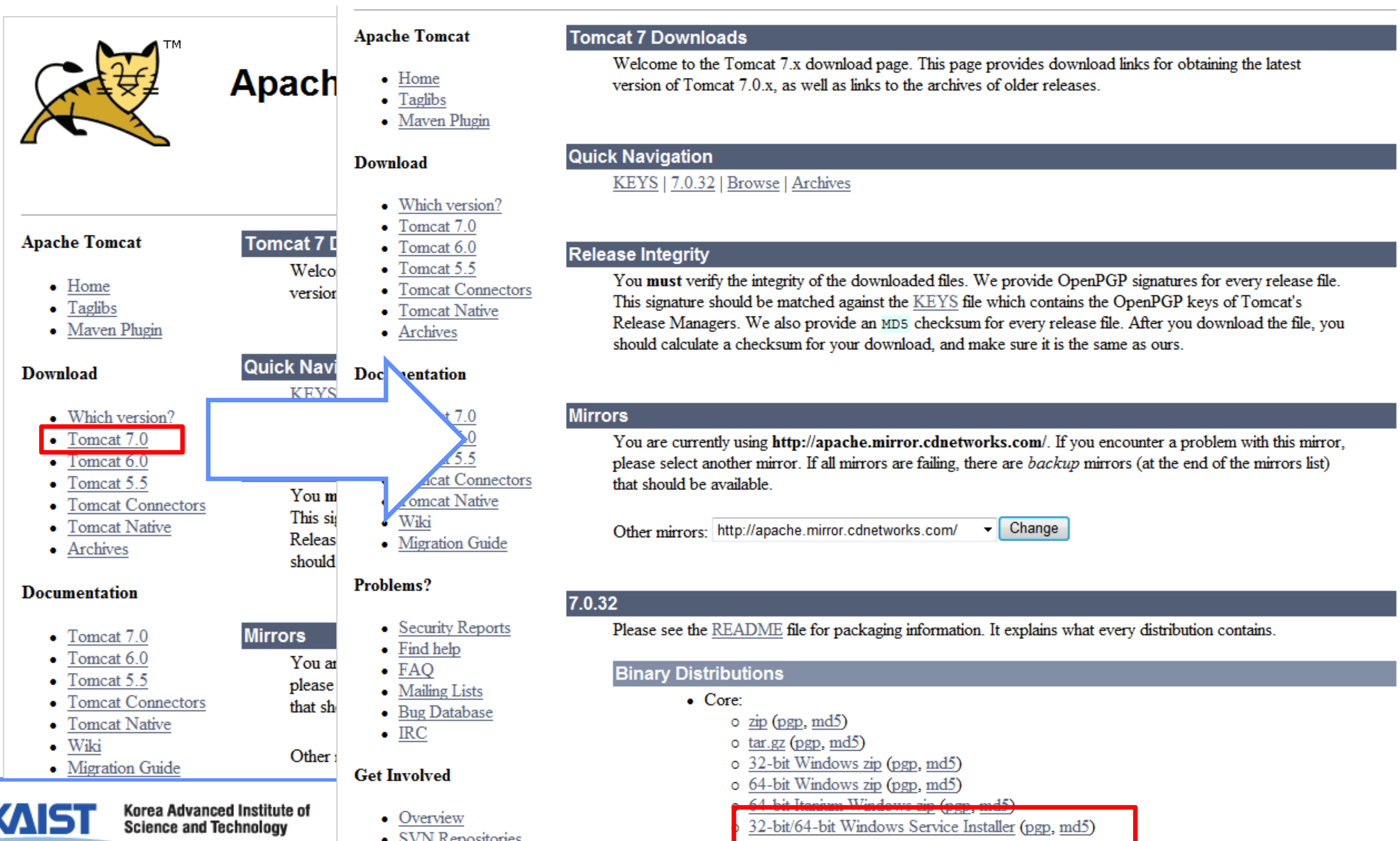
Install Web Server (Apache Tomcat)

- What is Tomcat?
  - Tomcat is an open source server from the Apache Software Foundation. It is a Web application server, which means that it comes ready to support programming using JavaServer Pages (JSPs) and servlets.

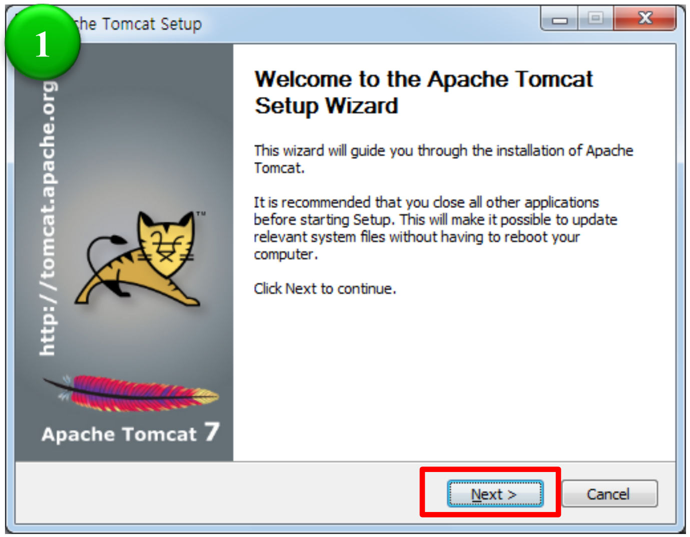
Download Apache Tomcat

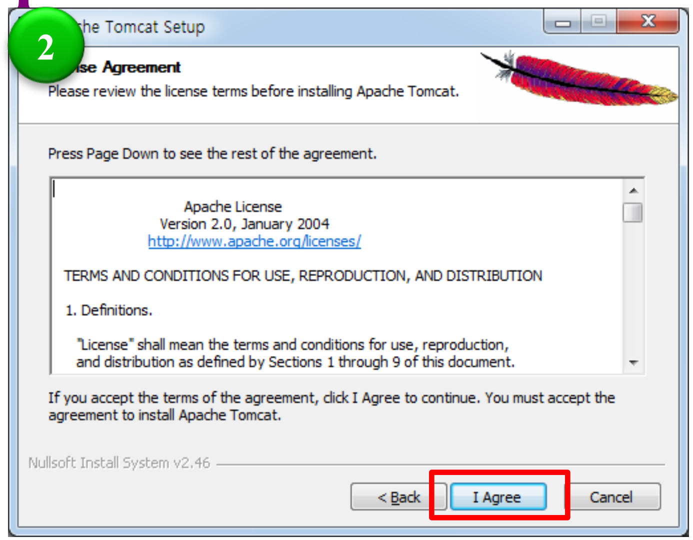
- http://tomcat.apache.org

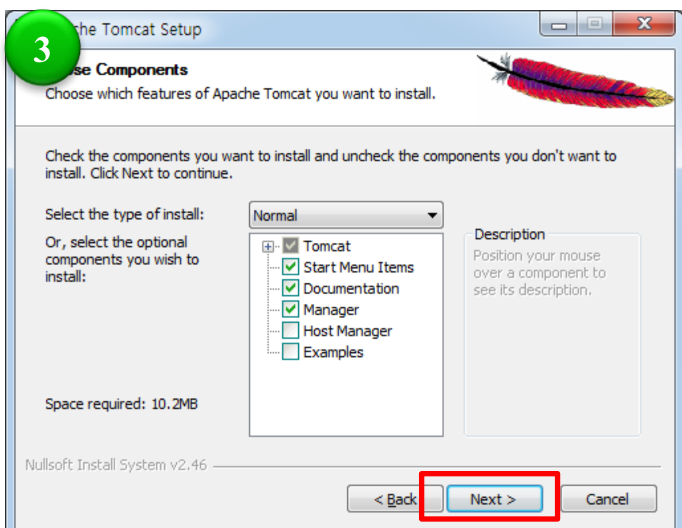
- Download :
  32-bit/64-bit Windows Service Installer

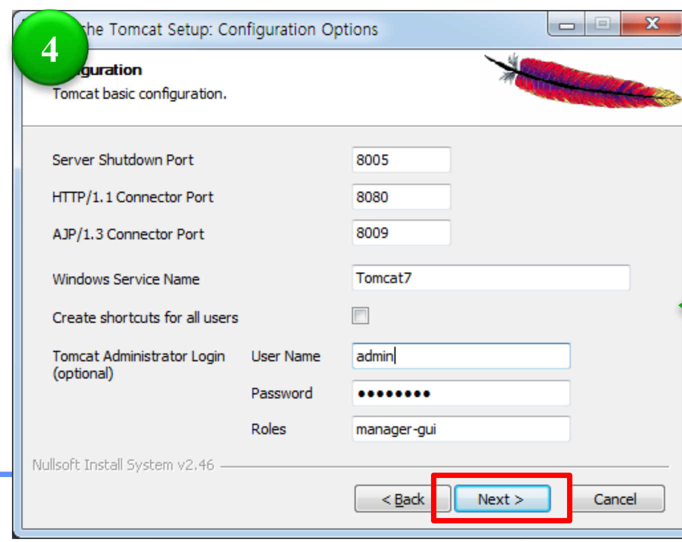


Install Apache Tomcat (1)



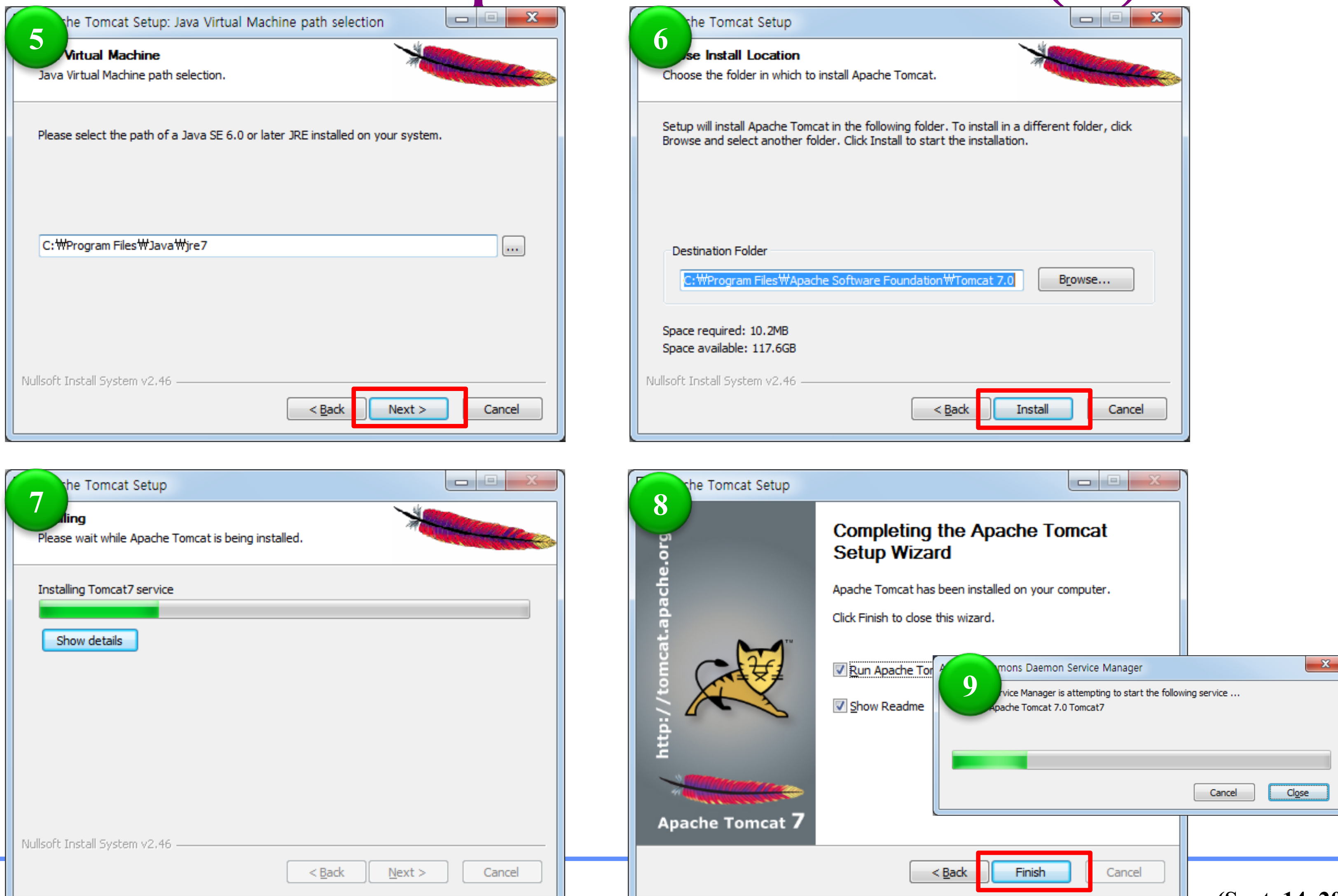






Enter User Name & Password

Install Apache Tomcat (2)



Install Apache Tomcat in Ubuntu

- sudo apt-get install openjdk-7-jre-headless
- sudo apt-get install tomcat7
- service tomcat7 start
- netstat -ntl

```
root@ubuntu:/usr# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::8080                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 127.0.0.1:8005          :::*                    LISTEN
```

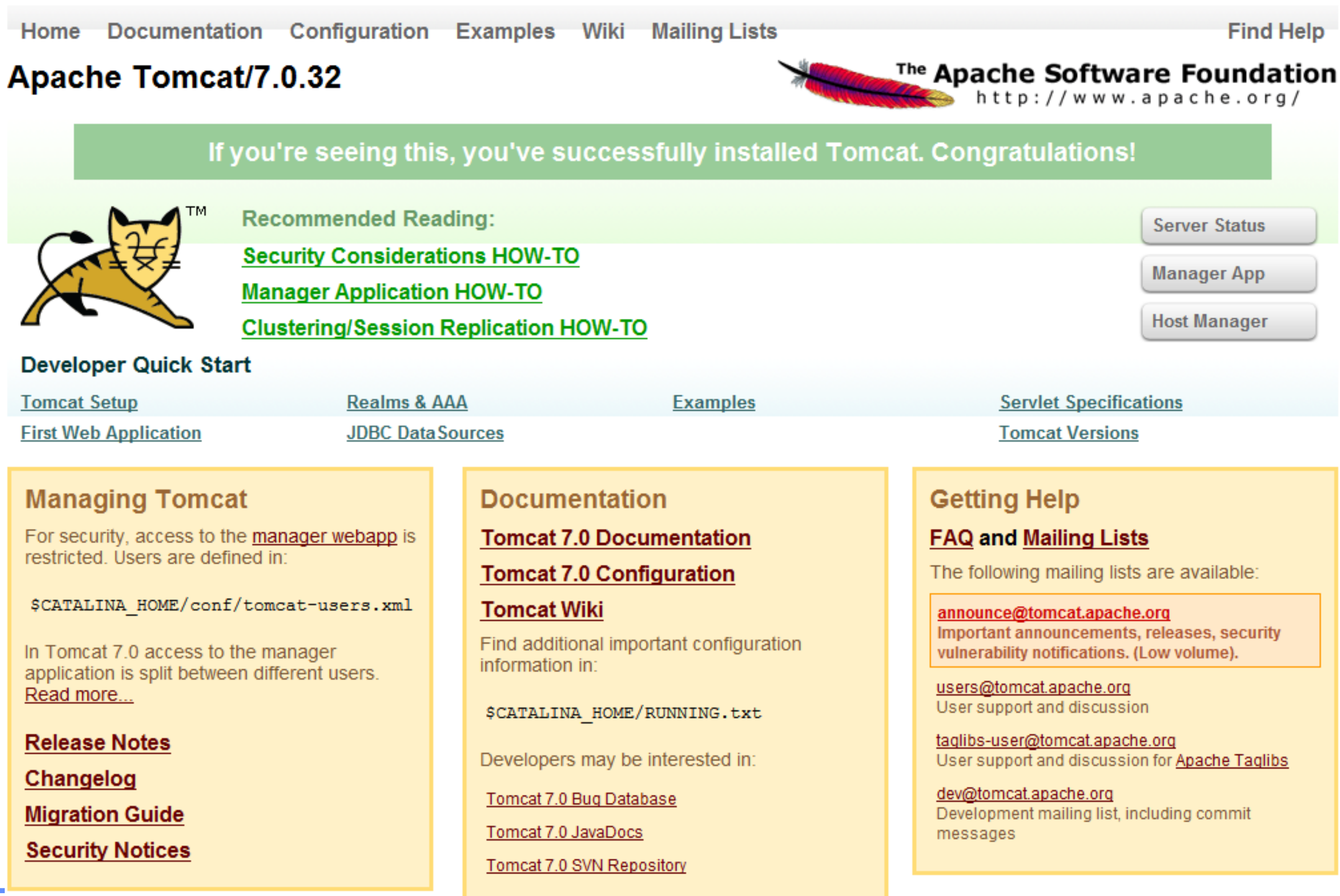
Install Apache Tomcat in Mac

- brew install cask java
- brew install tomcat
- /{TOMCAT HOME}/bin/startup.sh

```
root@ubuntu:/usr# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::8080                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 127.0.0.1:8005          :::*                    LISTEN
```

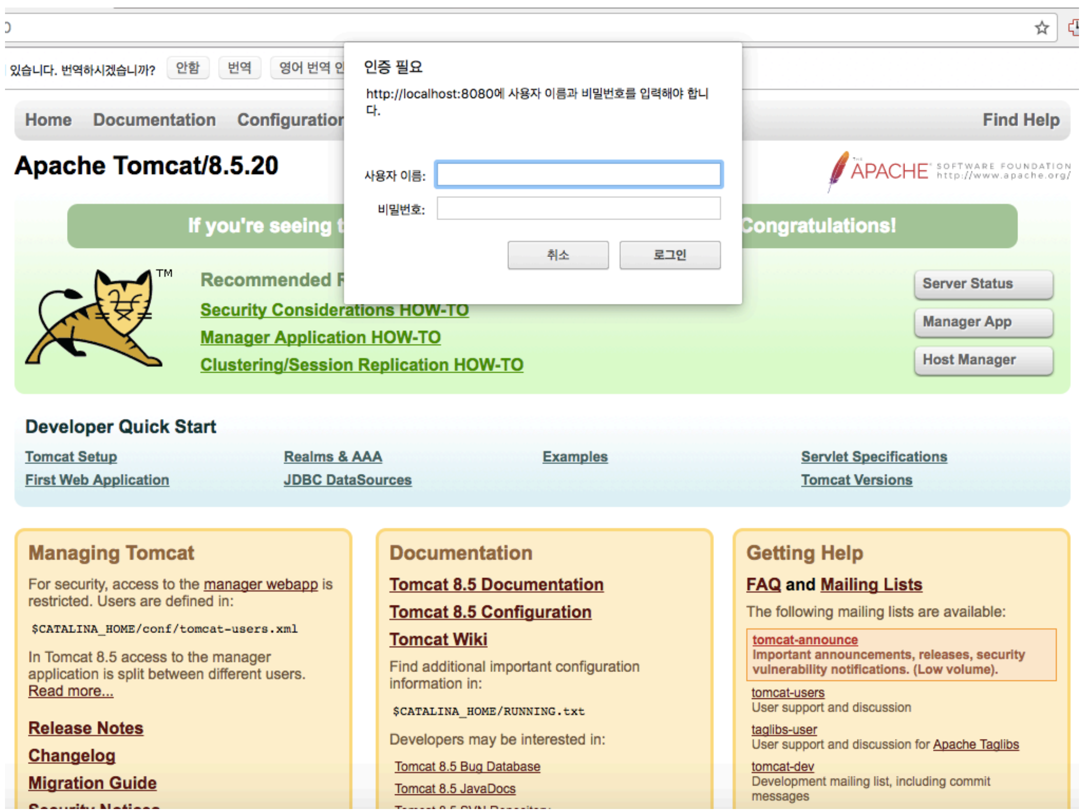
Management Web Page

- http://localhost:8080/ or http://[Host IP Address]:8080/



Authentication Problem

- User account should be created!



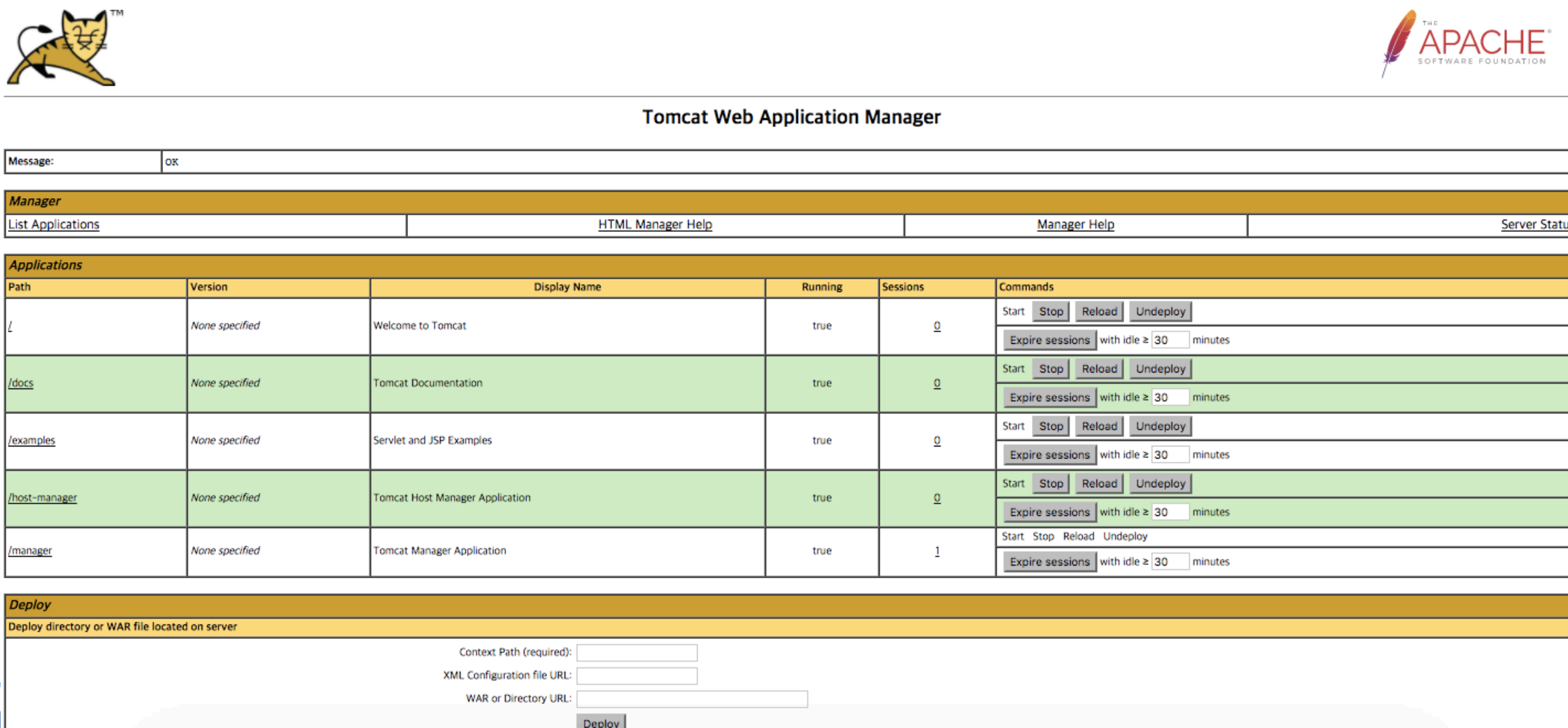
Create tomcat users

- Edit the *[/[TOMCAT HOME]/conf/tomcat-users.xml]()*
  - Location of TOMCAT HOME
  - Ubuntu: /etc/tomcat{version}/
  - Mac: /usr/local/Cellar/tomcat/{version}/libexec
- Add role named ‘manager-gui’
- Add user with your username and password with roles ‘manage-gui’
  - This user now can control GUI manager

```
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
<role rolename="manager-gui"/>
<user username="admin" password="1" roles="manager-gui"/>
</tomcat-users>
```

Manage Services

- We can life cycle of service (start /stop / restart / terminate)
  - There is already existing services including docs, examples, etc…

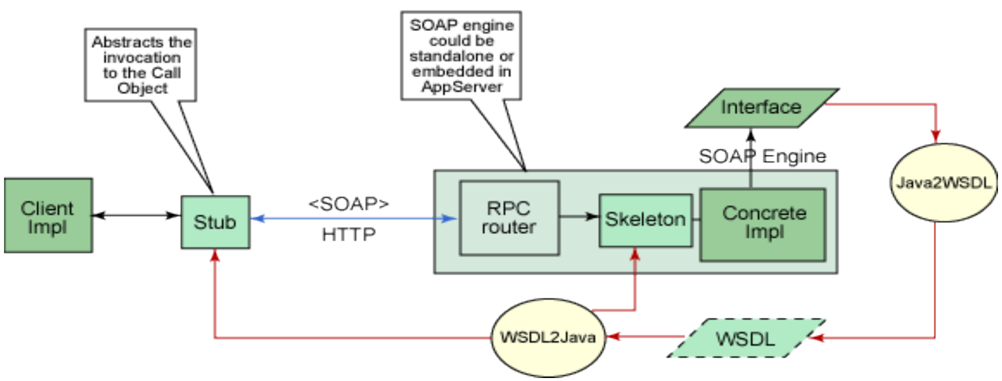


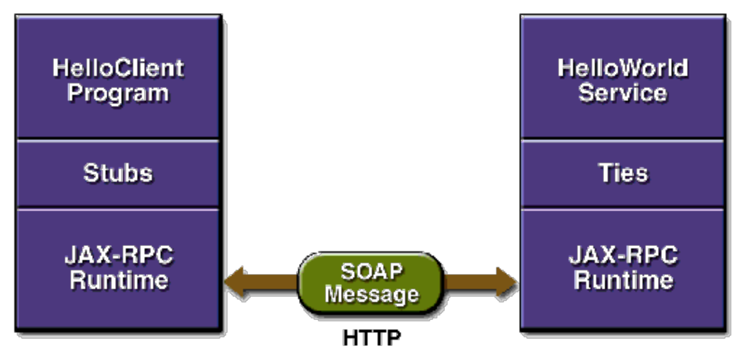
Example

- Chat example
  http://localhost:8080/examples/websocket/chat.xhtml
- Snake example
  http://localhost:8080/examples/websocket/snake.xhtml
- Draw example
  http://localhost:8080/examples/websocket/drawboard.xhtml
- Together
  http://143.248.xxx.xxx:8080/examples/websocket/chat.xhtml

RPC Example

- DemoRPCServer
  - Has the procedure “helloworld” that is called by clients
- DemoRPCClient
  - Calls, “helloworld”, the procedure which is located RPCServer
- Interfacing
  - XML-based WSDL(Web Service Description Language) is used as an IDL(interface Description Language)





Interface Description (similar to Skeleton)

```
package hello;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```

DemoRPCServer

```
package hello;
public class HelloImpl implements HelloIF {
    public String message = "Hello ";
    public String sayHello(String s) {
        return message + s;
    }
}
```

Jaxrpc-ri.xml (for war packaging)

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
   version="1.0"
   targetNamespaceBase="http://com.test/wsdl"
   typeNamespaceBase="http://com.test/types"
   urlPatternBase="/ws">


   <endpoint
      name="MyHello"
      displayName="HelloWorld Service"
      description="A simple web service"
      interface="hello.HelloIF"
      implementation="hello.HelloImpl"/>

   <endpointMapping
      endpointName="MyHello"
      urlPattern="/hello"/>


</webServices>
```

Deploy war into Tomcat server and WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location=
    "http://localhost:8080/hello-jaxrpc/hello?WSDL"
     packageName="hello"/>
</configuration>
```

DemoRPCClient

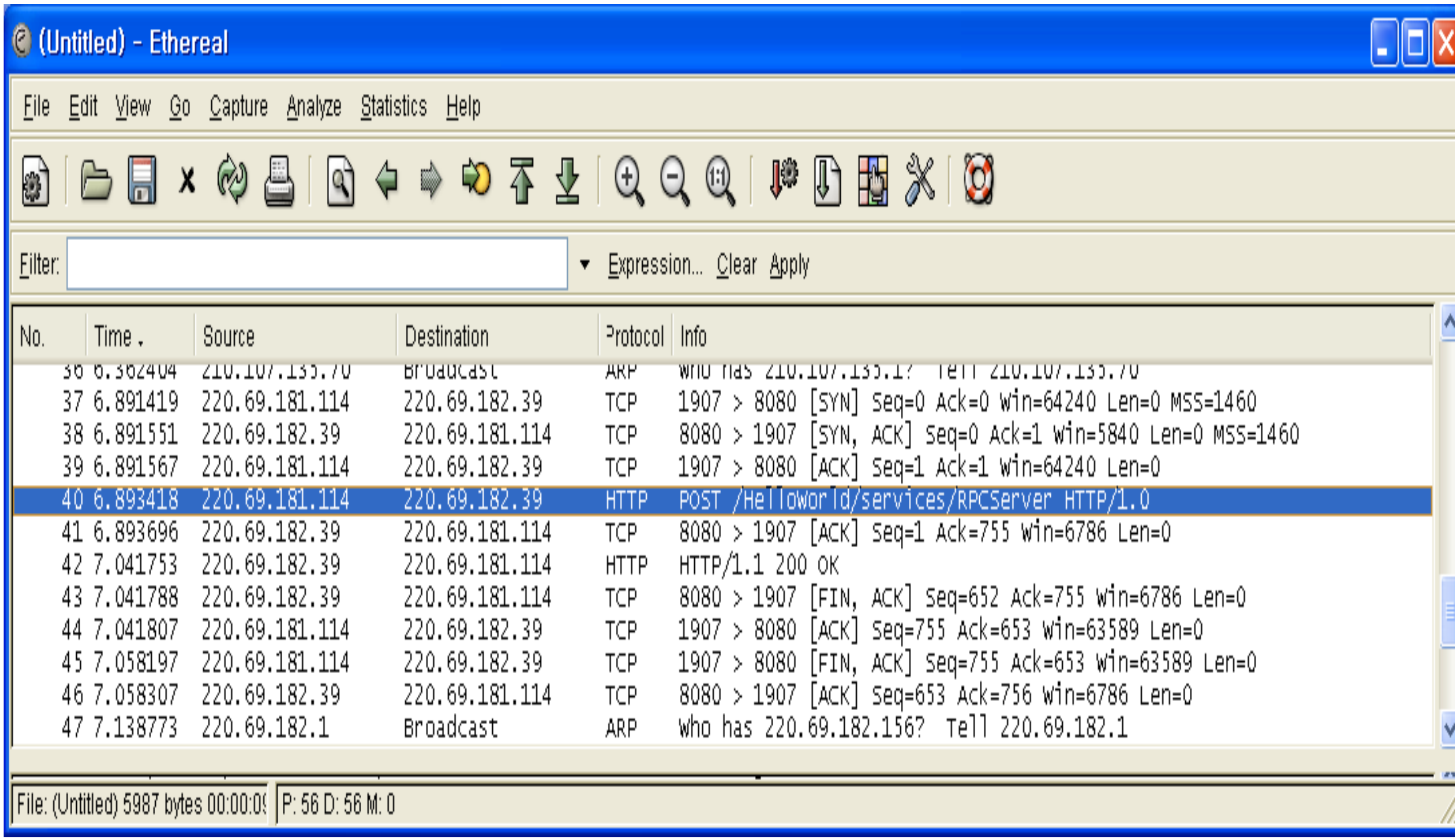
```
package hello;
import javax.xml.rpc.Stub;
public class HelloClient {
    public static void main(String[] args) {
        try {
            Stub stub = createProxy();
            HelloIF hello = (HelloIF)stub;
            System.out.println(hello.sayHello(”Foo"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    private static Stub createProxy() { // Note: MyHello_Impl is implementation-specific.
        return (Stub)(new MyHello_Impl().getHelloIFPort());
    }
}
```

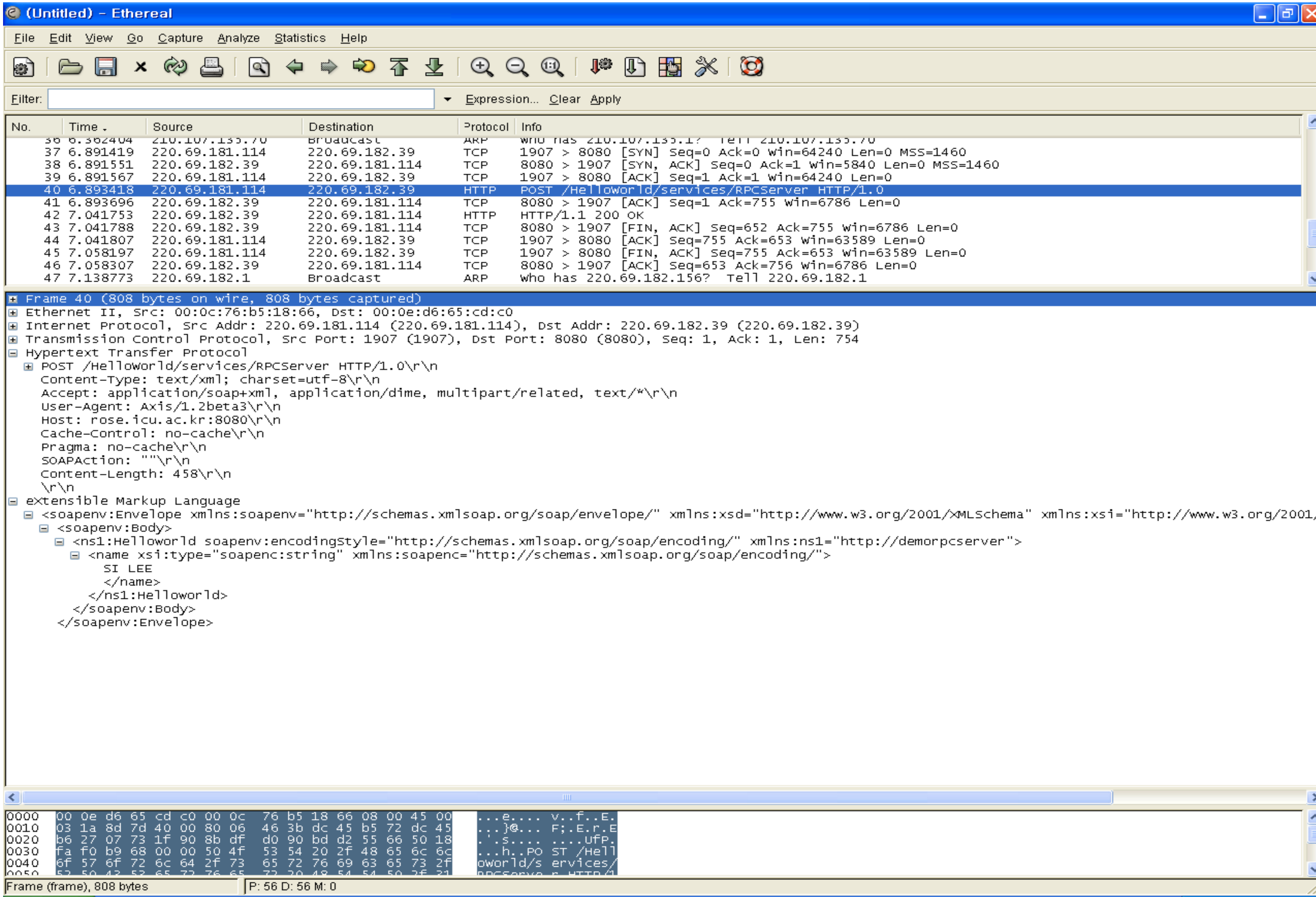
WSDL as an IDL

http://rose.icu.ac.kr:8080/HelloWorld/services/RPCServer?wsdl - Microsoft Internet Explorer


```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://demorpcserver" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://demorpcserver"
  xmlns:intf="http://demorpcserver" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <!--
      WSDL created by Apache Axis version: 1.2beta3
      Built on Aug 01, 2004 (05:59:22 PDT)
    -->
  - <wsdl:types>
    - <schema targetNamespace="http://demorpcserver" xmlns="http://www.w3.org/2001/XMLSchema">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      - <complexType name="ArrayOf_soapenc_string">
        - <complexContent>
          - <restriction base="soapenc:Array">
              <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:string[]" />
            </restriction>
          </complexContent>
        </complexType>
      </schema>
    </wsdl:types>
    <wsdl:message name="mainResponse" />
  - <wsdl:message name="HelloworldRequest">
      <wsdl:part name="name" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="mainRequest">
      <wsdl:part name="args" type="impl:ArrayOf_soapenc_string" />
    </wsdl:message>
  - <wsdl:message name="HelloworldResponse">
      <wsdl:part name="HelloworldReturn" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:portType name="RPCServer">
    - <wsdl:operation name="main" parameterOrder="args">
        <wsdl:input message="impl:mainRequest" name="mainRequest" />
        <wsdl:output message="impl:mainResponse" name="mainResponse" />
      </wsdl:operation>
    - <wsdl:operation name="Helloworld" parameterOrder="name">
        <wsdl:input message="impl:HelloworldRequest" name="HelloworldRequest" />
        <wsdl:output message="impl:HelloworldResponse" name="HelloworldResponse" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="RPCServerSoapBinding" type="impl:RPCServer">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="main">
        <wsdlsoap:operation soapAction="" />
      - <wsdl:input name="mainRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://demorpcserver" use="encoded" />
        </wsdl:input>
      - <wsdl:output name="mainResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://demorpcserver" use="encoded" />
        </wsdl:output>
```

Protocol of RPC




(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
37 6.891419 220.69.181.114 220.69.182.39 TCP 1907 > 8080 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
38 6.891551 220.69.182.39 220.69.181.114 TCP 8080 > 1907 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
39 6.891567 220.69.181.114 220.69.182.39 TCP 1907 > 8080 [ACK] Seq=1 Ack=1 Win=64240 Len=0
40 6.893418 220.69.181.114 220.69.182.39 HTTP POST /HelloWorld/services/RPCServer HTTP/1.0
41 6.893696 220.69.182.39 220.69.181.114 TCP 8080 > 1907 [ACK] Seq=1 Ack=755 Win=6786 Len=0
42 7.041753 220.69.182.39 220.69.181.114 HTTP HTTP/1.1 200 OK
43 7.041788 220.69.182.39 220.69.181.114 TCP 8080 > 1907 [FIN, ACK] Seq=652 Ack=755 Win=6786 Len=0
44 7.041807 220.69.181.114 220.69.182.39 TCP 1907 > 8080 [ACK] Seq=755 Ack=653 Win=63589 Len=0
45 7.058197 220.69.181.114 220.69.182.39 TCP 1907 > 8080 [FIN, ACK] Seq=755 Ack=653 Win=63589 Len=0
46 7.058307 220.69.182.39 220.69.181.114 TCP 8080 > 1907 [ACK] Seq=653 Ack=756 Win=6786 Len=0
47 7.138773 220.69.182.1 Broadcast ARP Who has 220.69.182.156? Tell 220.69.182.1
Frame 40 (808 bytes on wire, 808 bytes captured)
Ethernet II, Src: 00:0c:76:b5:18:66, Dst: 00:0e:d6:65:cd:c0
Internet Protocol, Src Addr: 220.69.181.114 (220.69.181.114), Dst Addr: 220.69.182.39 (220.69.182.39)
Transmission Control Protocol, Src Port: 1907 (1907), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 754
Hypertext Transfer Protocol
POST /HelloWorld/services/RPCServer HTTP/1.0\r\n
Content-Type: text/xml; charset=utf-8\r\n
Accept: application/soap+xml, application/dime, multipart/related, text/*\r\n
User-Agent: Axis/1.2beta3\r\n
Host: rose.icu.ac.kr:8080\r\n
Cache-Control: no-cache\r\n
Pragma: no-cache\r\n
SOAPAction: ""\r\n
Content-Length: 458\r\n
\r\n
eXtensible Markup Language
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
<soapenv:Body>
<ns1:HelloWorld soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://demorpcserver">
<name xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
SI LEE
</name>
</ns1:HelloWorld>
</soapenv:Body>
</soapenv:Envelope>
0000 00 0e d6 65 cd c0 00 0c 76 b5 18 66 08 00 45 00 ...e.... v..f..E.
0010 03 1a 8d 7d 40 00 80 06 46 3b dc 45 b5 72 dc 45 ...}@... F;.E.r.E
0020 b6 27 07 73 1f 90 8b df d0 90 bd d2 55 66 50 18 .'.s.... ....UfP.
0030 fa f0 b9 68 00 00 50 4f 53 54 20 2f 48 65 6c 6c ...h..PO ST /Hell
0040 6f 57 6f 72 6c 64 2f 73 65 72 76 69 63 65 73 2f oWorld/s ervices/
Frame (frame), 808 bytes
P: 56 D: 56 M: 0

gRPC and Interface Definition

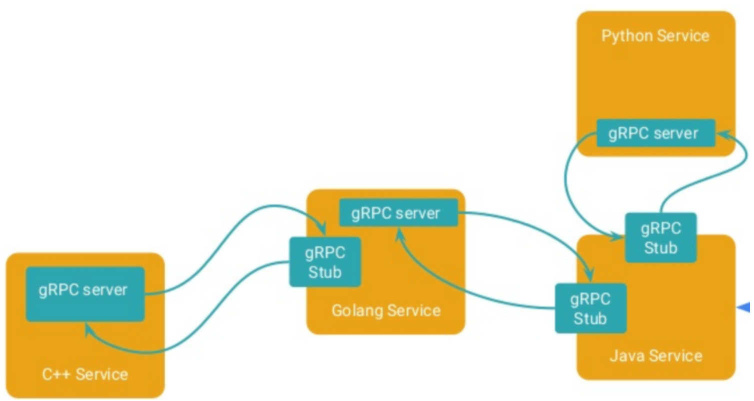
- gRPC uses Protocol Buffers as the **Interface Definition Language (IDL)** for describing both the **service interface** and the **structure of the payload messages**.
- gRPC provides Protocol Compiler plugins that generate Client- and Server-side APIs with an interface definition in a .proto file.

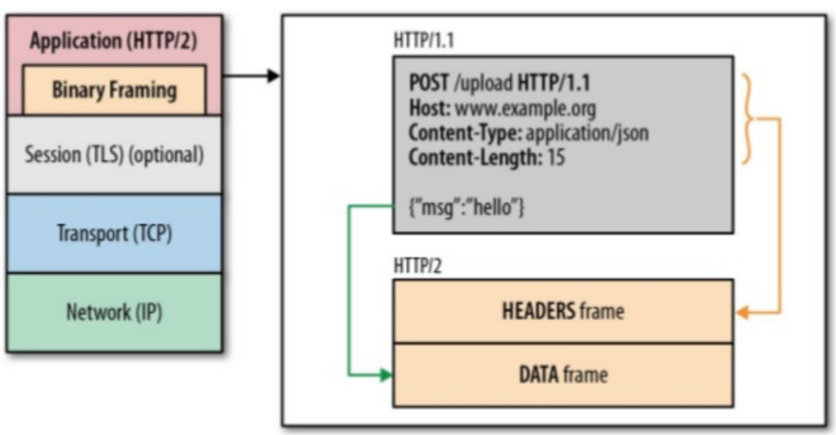
Example) Bidirectional Streaming RPC Definition

```
service RouteGuide {
  rpc RouteChat (stream RouteNote) returns (stream RouteNote);
  ...
}

message RouteNote {
  string location = 1;
  string message = 2;
}
```

gRPC over HTTP/2

- The abstract protocol defined above is implemented over HTTP/2.
  - gRPC bidirectional streams are mapped to HTTP/2 streams.
  - The contents of Call Header and Initial Metadata are sent as HTTP/2 headers and subject to HPACK compression.
  - Payload Messages are serialized into a byte stream of length prefixed gRPC frames which are then fragmented into HTTP/2 frames at the sender and reassembled at the receiver.
  - Status and Trailing-Metadata are sent as HTTP/2 trailing headers.
- gRPC inherits the flow control mechanisms in HTTP/2 and uses them to enable fine-grained control of the amount of memory used for buffering in-flight messages.



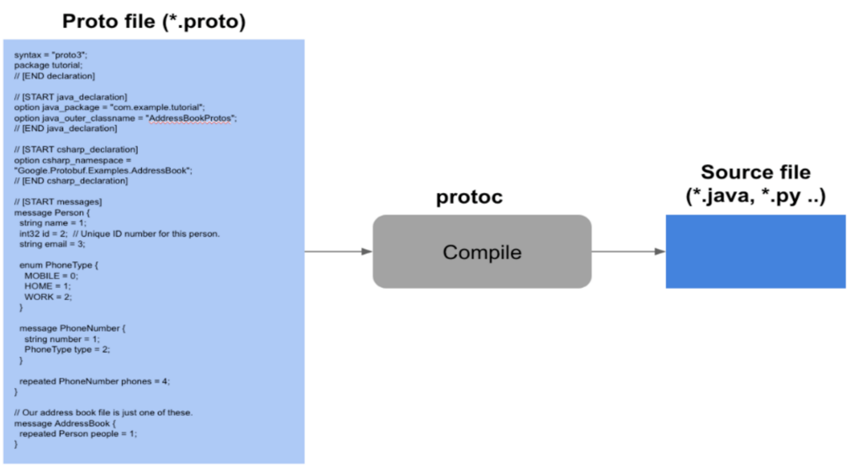


Protocol Buffer

- The Prototype Buffer is a serialized data structure developed by Google and released as open source.
- Serialization is the act of storing data in a binary stream for storage in a file or for transmission over a network.
- It supports various languages such as C ++, C #, Go, Java, Python, Object C, Javascript, Ruby, etc. Serialization speed is fast and serialized file size is small.

Protobuf Compile

- To use the protocol buffer, define the data type to be stored as proto file. Define a datatype that has no dependency on a particular language.
- When you compile the proto file with the protoc compiler, you create a data class file of the appropriate type for each language.



<An example of compiling with Python>

protoc -I=./ --python_out=./ ./address.proto

```python
import address_pb2

person = address_pb2.Person()

person.name = 'Terry'
person.age = 42
person.email = 'terry@mycompany.com'

f = open('myaddress','wb')
f.write(person.SerializeToString())
f.close()
```

Protobuf to JSON

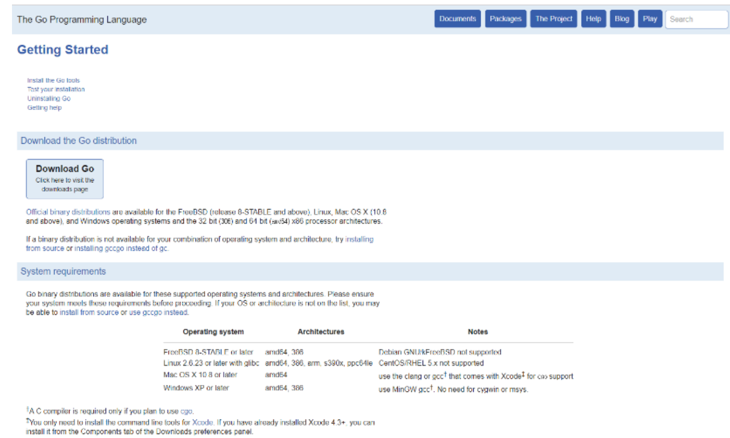
- When implementing a REST API such as HTTP / JSON from a client (mobile) to a server, it is necessary to serialize JSON into protocol buffer format before transmission, to transmit the packet with a reduced overall packet amount.
- API gateway is placed in front of the back-end server, and the message body that comes in the protocol buffer is converted into JSON and passed to the back-end API server.

```
from google.protobuf.json_format import MessageToJson
jsonObj = MessageToJson(person)
print jsonObj
```

→

```
{
 "age": 42,
 "name": "Terry",
 "email": "terry@mycompany.com"
}
```

gRPC Environment Setting (Linux)



- **Golang Installation**
  – Install Golang binaries
  – https://golang.org/doc/install?cm_mc_uid=02652595810614900694309&cm_mc_sid_50200000=1490500863
  – Install the downloaded compressed file through the following command

```
tar -C /usr/local -xzf <다운로드 받은 압축파일>
```

  – Setting Environment Variables in $ HOME / .profile

```
$HOME 아래에 work라는 디렉토리 생성

mkdir $HOME/work
export GOROOT=/usr/local/go
export GOPATH=$HOME/work
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

gRPC Environment Setting (Linux)

- **Check Golang Execution**
  - hello.go

```
Package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

  - Build source code

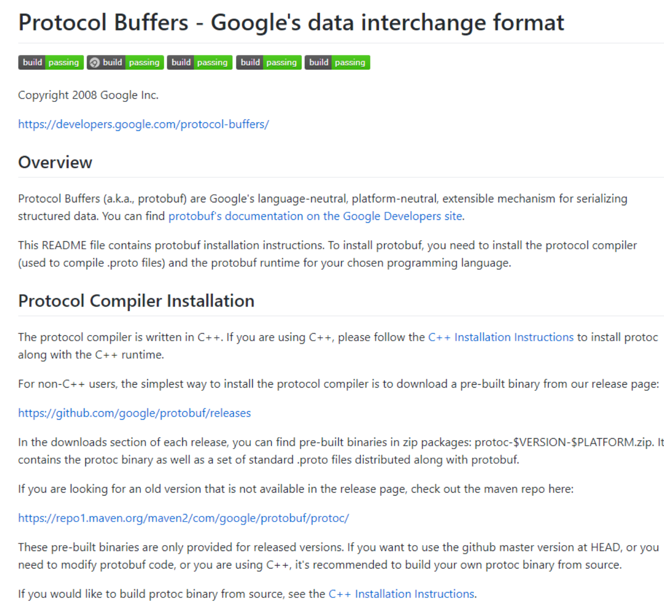
```
cd $GOPATH/src/hello
go build
```

  - Execute

```
./hello
hello, world
```

Install Protocol Buffer

- Download Protocol Buffer Archive
  - https://github.com/google/protobuf/releases


Protocol Buffers - Google's data interchange format

build passing | build passing | build passing | build passing | build passing

Copyright 2008 Google Inc.

https://developers.google.com/protocol-buffers/

Overview

Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. You can find protobuf's documentation on the Google Developers site.

This README file contains protobuf installation instructions. To install protobuf, you need to install the protocol compiler (used to compile .proto files) and the protobuf runtime for your chosen programming language.

Protocol Compiler Installation

The protocol compiler is written in C++. If you are using C++, please follow the C++ Installation Instructions to install protoc along with the C++ runtime.

For non-C++ users, the simplest way to install the protocol compiler is to download a pre-built binary from our release page:

https://github.com/google/protobuf/releases

In the downloads section of each release, you can find pre-built binaries in zip packages: protoc-$VERSION-$PLATFORM.zip. It contains the protoc binary as well as a set of standard .proto files distributed along with protobuf.

If you are looking for an old version that is not available in the release page, check out the maven repo here:

https://repo1.maven.org/maven2/com/google/protobuf/protoc/

These pre-built binaries are only provided for released versions. If you want to use the github master version at HEAD, or you need to modify protobuf code, or you are using C++, it's recommended to build your own protoc binary from source.

If you would like to build protoc binary from source, see the C++ Installation Instructions.

- Unarchive and register to $PATH

```
#### ProtoBuf ###
export PATH=$PATH:/Users/mjkong/Dev/sdk/protoc-3.2.0rc2-osx-x86_64/bin
```

- Install Protobuf Golang plugin

```
go get -u github.com/golang/protobuf/{proto,protoc-gen-go}
```

gRPC Example

- **Hello world example**

```
cd
$GOPATH/src/google.golang.org/grpc/helloworld
```

  - [dir] greeter_client – Client code
  - [dir] greeter_server – Server code
  - [dir] helloworld - protobuf file specify the gRPC service. With command ‘protoc’ .pb.go file will be created and it specify the message type of server-client communication
  - Run Server Code

```
go run greeter_server/main.go
```

  - Run Client Code

```
go run greeter_client/main.go
```

  - Check out message “Greeting: Hello world”

gRPC Example

- **Service Update**

```
$GOPATH/src/google.golang.org/grpc/examples/helloworld/helloworld/helloworld.proto
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

  – Build gRPC Code

```
protoc -I helloworld/ helloworld/helloworld.proto --go_out=plugins=grpc:helloworld
```

gRPC Example

- **Service Update**
  - Server code updated

```go
func (s *server) SayHelloAgain(ctx context.Context, in
*pb.HelloRequest) (*pb.HelloReply, error) {
    return &pb.HelloReply{Message: "Hello again " +
in.Name}, nil
}
```

  - Client code updated

```go
r, err = c.SayHelloAgain(context.Background(),
&pb.HelloRequest{Name: name})
if err != nil {
    log.Fatalf("could not greet: %v", err)
}
log.Printf("Greeting: %s", r.Message)
```

  - Execution Result

```
mjmac:helloworld mjkong$ go run greeter_client/main.go
2017/03/26 22:22:28 Greeting: Hello world
2017/03/26 22:22:28 Greeting: Hello again world
```

Node.js gRPC Example (Protobuf)

```
service BookService {
  rpc List (Empty) returns (BookList) {}
  rpc Insert (Book) returns (Empty) {}
  rpc Get (BookIdRequest) returns (Book) {}
  rpc Delete (BookIdRequest) returns (Empty) {}
  rpc Watch (Empty) returns (stream Book) {}
}

message Empty {}

message Book {
  int32 id = 1;
  string title = 2;
  string author = 3;
}

message BookList {
  repeated Book books = 1;
}
message BookIdRequest {
  int32 id = 1;
}
```

Node.js gRPC Example (Server)

```
var grpc = require('grpc');

var booksProto = grpc.load('books.proto');

var server = new grpc.Server();


server.bind('0.0.0.0:50051',
  grpc.ServerCredentials.createInsecure());
console.log('Server running at
http://0.0.0.0:50051');
server.start();
```

```
server.addService(booksProto.books.BookService.service, {
    list: function(call, callback) {
        callback(null, books);
    },
    insert: function(call, callback) {
        var book = call.request;
        books.push(book);
        callback(null, {});
    },
    get: function(call, callback) {
        for (var i = 0; i < books.length; i++)
            if (books[i].id == call.request.id)
                return callback(null, books[i]);
        callback({
            code: grpc.status.NOT_FOUND,
            details: 'Not found'
        });
    },
    delete: function(call, callback) {
        for (var i = 0; i < books.length; i++) {
            if (books[i].id == call.request.id) {
                books.splice(i, 1);
                return callback(null, {});
            }
        }
        callback({
            code: grpc.status.NOT_FOUND,
            details: 'Not found'
        });
    }
});
```

Node.js gRPC Example (Client)

```
var grpc = require('grpc');

var booksProto = grpc.load('books.proto');

var client = new
booksProto.books.BookService('127.0.0.1:50051',
  grpc.Credentials.createInsecure());

function printResponse(error, response) {
  if (error)
    console.log('Error: ', error);
  else
    console.log(response);
}

function listBooks() {
  client.list({}, function(error, books) {
    printResponse(error, books);
  });
}
```

```
function insertBook(id, title, author) {
  var book = { id: parseInt(id), title: title,
author: author };
  client.insert(book, function(error, empty) {
    printResponse(error, empty);
  });
}

function getBook(id) {
  client.get({ id: parseInt(id) },
function(error, book) {
    printResponse(error, book);
  });
}

function deleteBook(id) {
  client.delete({ id: parseInt(id) },
function(error, empty) {
    printResponse(error, empty);
  });
}
```

Result

```
$ node client.js insert 2 "The Three Musketeers"
"Alexandre Dumas"
{}

$ node client.js list
{ books:
   [ { id: 123,
       title: 'A Tale of Two Cities',
       author: 'Charles Dickens' },
     { id: 2,
       title: 'The Three Musketeers',
       author: 'Alexandre Dumas' } ] }
```

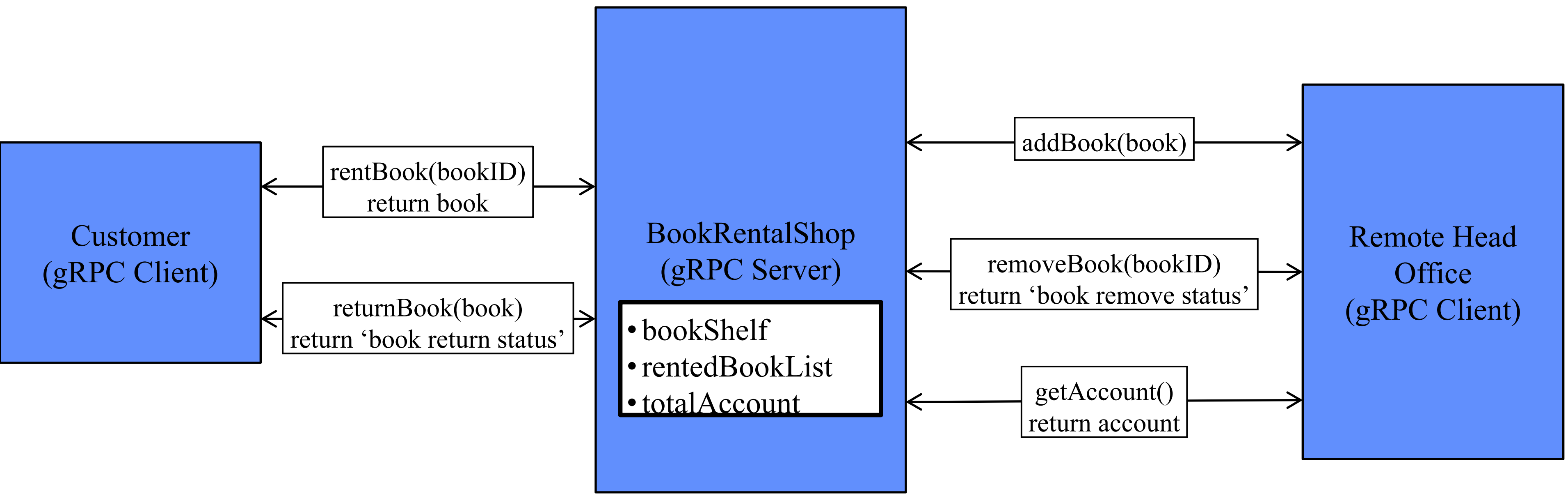
```
$ node client.js delete 123
{}

$ node client.js list
{ books:
   [ { id: 2,
       title: 'The Three Musketeers',
       author: 'Alexandre Dumas' } ] }

$ node client.js get 2
{ id: 2,
  title: 'The Three Musketeers',
  author: 'Alexandre Dumas' }
```

Lab 5. RPC Programming

- Read text materials and test practices
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Objectives
  - Understand the process of RPC
  - Learn the how to analyze network packet using wireshark
  - Understand packet structure of RPC

Lab Requirements

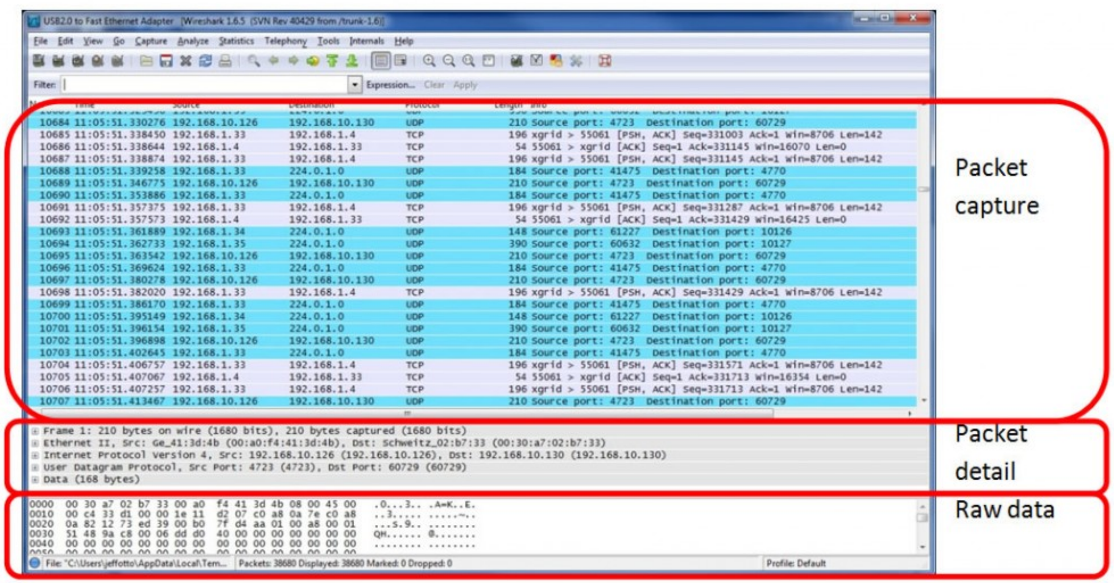
1. Follow the steps to deploy and call the gRPC service with another project names
2. Modify gRPC service to implement book store



* Add $2 per rent to account
* Handle the exceptions within the acceptable range

Lab. Analyzing RPC packets with Wireshark

- Prerequisite) WinPcap is the library for capturing packets and looking network status. WinPcap can be downloaded in following address. http://www.winpcap.org/install/default.htm
- We can also download Wireshark in https://www.wireshark.org/download.html



Service Senario

- Service Health Check Interface

```
syntax = "proto3";

package servicestatus;

service HealthCheck {
  rpc Status (StatusRequest) returns
(HealthCheckResult) {}
}

message StatusRequest {

}

message HealthCheckResult {
  string Status = 1;
}
```

RPC Service Interface Definition (Protobuf)

```
var hello_proto = grpc.load(PROTO_PATH).helloworld;

var servicestatus_proto = grpc.load(__dirname +
"/servicestatus.proto").servicestatus;
function sayHello(call, callback) {
  callback(null, {message: 'Hello ' + call.request.name});
}

function statusRPC(call, callback) {
  console.log("statusRPC", call);
  callback(null, {Status: 'MagicResponseCodeOK'});
}

function main() {
  var server = new grpc.Server();
  server.addProtoService(hello_proto.Greeter.service, {sayHello:
sayHello});
  server.addProtoService(servicestatus_proto.HealthCheck.service, {
status: statusRPC });
  server.bind('0.0.0.0:50051',
grpc.ServerCredentials.createInsecure());
  server.start();
}
```

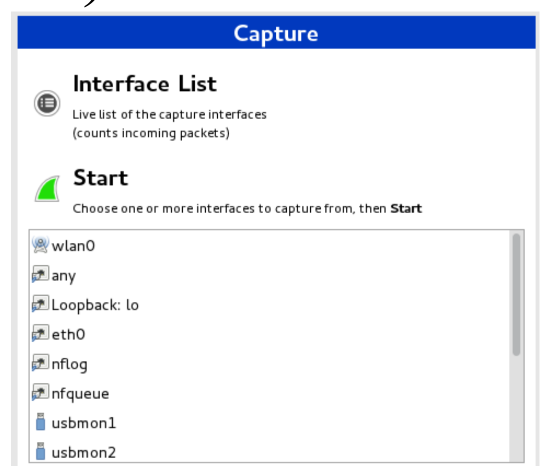
Server Stub

```
var servicestatus_proto =
grpc.load(PROTO_PATH).servicestatus;

function main() {
  var client = new
servicestatus_proto.HealthCheck('localhost:50051',
grpc.credentials.createInsecure());
  client.status({}, function(err, response) {
    console.log('Greeting:', response);
  });
}
```

Client Stub

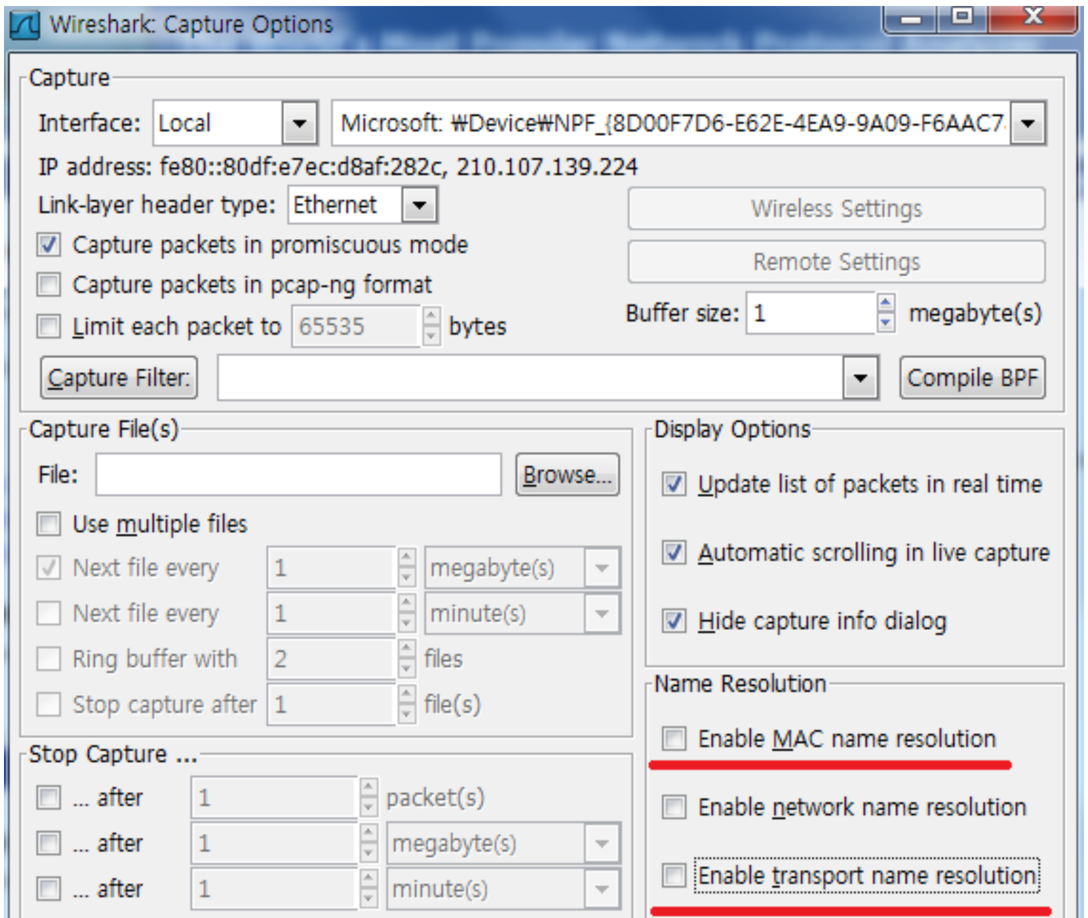
RPC operation (1)

- Identifying RPC operation with Wireshark
- After starting Wireshark, you can find a button which is in above red box. Click the button.
- Select the appropriate network interface (eth0, eth1, wifi, virtual bridge and etc.)



RPC operation (2)

- Complete the following configuration as shown.



RPC operation (3)

- After configuration, we can see the RPC operation results.

2	0…	127.0.0.1	127.0.0.1	TCP	68	50051 → 53109 [SYN, ACK] Seq=0 A
3	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=1 Ack=1
4	0…	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 50051 → 5310
5	0…	127.0.0.1	127.0.0.1	HTTP2	120	Magic, SETTINGS, WINDOW_UPDATE
6	0…	127.0.0.1	127.0.0.1	HTTP2	71	SETTINGS
7	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=16 Ack=6
8	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=65 Ack=1
9	0…	127.0.0.1	127.0.0.1	HTTP2	69	WINDOW_UPDATE
10	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=65 Ack=2
11	0…	127.0.0.1	127.0.0.1	HTTP2	330	HEADERS, DATA
12	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=29 Ack=3
13	0…	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
14	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=29 Ack=3
15	0…	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
16	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=348 Ack=
17	0…	127.0.0.1	127.0.0.1	HTTP2	244	HEADERS, DATA, HEADERS
18	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=348 Ack=

RPC operation (4)

- HTTP/2 typed gRPC packets
  - Payload is encoded in protobuf serialization

```
▼ HyperText Transfer Protocol 2
  ▼ Stream: Magic
      Magic: PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
  ▼ Stream: SETTINGS, Stream ID: 0, Length 18
      Length: 18
      Type: SETTINGS (4)
    ▶ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
    ▶ Settings - Enable PUSH : 0
    ▶ Settings - Max concurrent streams : 0
    ▶ Settings - Initial Windows size : 65535
  ▼ Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
      Length: 4
      Type: WINDOW_UPDATE (8)
    ▶ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 1111 0000 0000 0000 0001 = Window Size Increment: 983041
```

```
▼ HyperText Transfer Protocol 2
  ▼ Stream: HEADERS, Stream ID: 1, Length 251
      Length: 251
      Type: HEADERS (1)
    ▶ Flags: 0x04
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Header Block Fragment: 40073a736368656d6504687474704007 3a6d6574686f6404...
      [Header Length: 296]
    ▶ Header: :scheme: http
    ▶ Header: :method: POST
    ▶ Header: :path: /servicestatus.HealthCheck/Status
    ▶ Header: :authority: localhost:50051
    ▶ Header: grpc-encoding: identity
    ▶ Header: grpc-accept-encoding: deflate,gzip
    ▶ Header: te: trailers
    ▶ Header: content-type: application/grpc
    ▶ Header: user-agent: grpc-node/0.11.1 grpc-c/0.12.0.0 (osx)
      Padding: <MISSING>
```

Protobuf serialization

```
{
    "userName": "Martin",
    "favouriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

JSON

```
message Person {
    required string user_name       = 1;
    optional int64  favourite_number = 2;
    repeated string interests       = 3;
}
```

Protobuf Schema