

WebServices & WebSockets

Jakub Daněk (KIV ZČU, Yoso Czech s.r.o.)

WebService and WebSocket **basics**

<http://www.danekja.org> (Twitter & LinkedIn links)

<http://www.yoso.fi> (yep, in Finnish)

Motivation

- **How to connect different applications?**
 - Written in different languages
 - Deployed on different platforms
 - By different people

WebService

- **W3C definition:**

A **Web service** is a software system designed to support **interoperable machine-to-machine interaction over a network**. It has an interface described in a **machine-processable format** (specifically WSDL).

Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

- Restricted to SOAP, not meant as generic definition (other things can be webservices too...)

WebService

- **Way to connect applications on web**

- Using standardized open protocols (HTTP, SOAP)
- Using standardized open data formats (XML, JSON)

→ providers do not need to care about their client's technology stack, platform, etc.

WebService

- **Example**

```
GET /posts HTTP/1.1
```

```
//response
```

```
HTTP/1.1 200 OK
```

```
{  
  "id": 4,  
  "content": "Good morning, we've talked the whole night  
              through.",  
  "createdBy": "Duelling Cavalier"  
}
```

WebService

- **Black box**

- Consumers are not aware of implementation of the services they are consuming

- **Stateless**

- In most cases web service calls are stateless
- Simpler design
- Better scalability

WebService - Uses

- **Reusable components**

- User authentication, weather forecast, currency conversion, navigation

- **Connect existing software**

- Independent of programming language (web standards)
- Independent of platform (Unix, Windows, Raspberry PI)

Sources

- [**http://www.w3schools.com/xml/xml_services.asp**](http://www.w3schools.com/xml/xml_services.asp)
- [**https://www.w3.org/TR/ws-arch/**](https://www.w3.org/TR/ws-arch/)

Simple Object Access Protocol (SOAP)

SOAP

- **Messaging protocol**

- Message format
- Encoding rules for datatypes

- **XML-based**

- All messages exchanged in XML format

- **Actions determined by message content**

SOAP - Message

- **Envelope**

- XML element encapsulating the SOAP message

- **Header**

- Means for protocol extension, may provide additional information about the message sent, security, etc.

- **Body**

- The message data

- **Fault**

- Information generated by a processing node on error

SOAP

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-  
envelope">  
  <env:Fault>  
    ...  
  </env:Fault>  
  <env:Header>  
    ...  
  </env:Header>  
  <env:Body>  
    ...  
  </env:Body>  
</env:Envelope>
```

SOAP

- **Advantages**

- Flexible
- Extensible
- Uses XML – XSD, internationalization, namespaces

SOAP

- **Disdvantages**

- No standardized API design approach
- Uses XML
 - Message size and complexity grows quickly
 - Performance issues

SOAP - Example 1 (request)

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
 soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

 <soap:Body xmlns:m="http://www.example.org/stock">

 <m:GetStockPrice>

 <m:StockName>IBM</m:StockName>

 </m:GetStockPrice>

 </soap:Body>

</soap:Envelope>

SOAP - Example 1 (response)

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.example.org/stock">
```

```
    <m:GetStockPriceResponse>
```

```
      <m:Price>34.5</m:Price>
```

```
    </m:GetStockPriceResponse>
```

```
  </soap:Body>
```

```
</soap:Envelope>
```


SOAP - Example 2

- **Example**

See the attached file **WebServices_sovohje_en.pdf**

- **Portion of WS API specification of OP Bank (Finland)**
- **Specifically pages 12 and 13**

WebService Description Language (WSDL)

- **XML Schema for describing WebServices**
- **Not restricted to SOAP**
 - **But commonly used in conjunction**
- **Allows generating of client code**

WSDL - Elements

- **<types>**
 - defines the (XML Schema) data types used by the web service
- **<message>**
 - defines the data elements for each operation
- **<portType>**
 - operations that can be performed and the messages involved as input, output, on error
- **<binding>**
 - binding of portType to certain protocol (e.g. SOAP)

WSDL Example – OP Bank (www.pohjola.fi) WS API

WSDL - Elements

- **<types>**
- **defines the (XML Schema) data types used by the web service**
- **standard XSD**

<wsdl:types>

```
<xsd:schema targetNamespace="http://mlp.op.fi/OPCertificateService"
  elementFormDefault="qualified" attributeFormDefault="qualified">
```

```
  <xsd:complexType name="CertificateRequestHeader">
```

```
    ...
```

```
  </xsd:complexType>
```

```
  <xsd:complexType name="GetCertificateRequest">
```

```
    <xsd:sequence>
```

```
      <xsd:element name="RequestHeader" type="tns:CertificateRequestHeader" nillable="false"/>
```

```
      <xsd:element name="ApplicationRequest" type="xsd:base64Binary" nillable="false"/>
```

```
    </xsd:sequence>
```

```
  </xsd:complexType>
```

```
  <xsd:element name="getCertificatein" type="tns:GetCertificateRequest"/>
```

```
</xsd:schema>
```

```
</wsdl:types>
```

WSDL - Elements

- **<message>**
 - **defines the data elements for each operation**

```
<wsdl:message name="getCertificateRequest">  
  <wsdl:part element="tns:getCertificatein" name="getCertificatein"/>  
</wsdl:message>
```

WSDL - Elements

- **<portType>**
 - **operations that can be performed and the messages involved as input, output, on error**

```
<wsdl:portType name="OPCertificateServicePortType">
```

```
  <wsdl:operation name="getCertificate">
```

```
    <wsdl:input message="tns:getCertificateRequest" name="getCertificateRequest"/>
```

```
    <wsdl:output message="tns:getCertificateResponse" name="getCertificateResponse"/>
```

```
    <wsdl:fault message="tns:certificateServiceFault" name="certificateServiceFault"/>
```

```
  </wsdl:operation>
```

```
</wsdl:portType>
```

WSDL - Elements

- **<binding>**
- **binding of portType to certain protocol (e.g. SOAP)**

```
<wsdl:binding name="OPCertificateServiceHttpBinding" type="tns:OPCertificateServicePortType">  
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>  
  <wsdl:operation name="getCertificate">  
    <soap:operation soapAction=""/>  
  
    <wsdl:input name="getCertificateRequest"> <soap:body use="literal"/> </wsdl:input>  
  
    <wsdl:output name="getCertificateResponse"> <soap:body use="literal"/> </wsdl:output>  
  
    <wsdl:fault name="certificateServiceFault"> <soap:fault use="literal"/> </wsdl:fault>  
  </wsdl:operation>  
</wsdl:binding>
```


Sources

- <https://www.w3.org/TR/2007/REC-soap12-part0-2-0070427/>
- <https://www.w3.org/TR/soap12-part1/>
- <https://www.pohjola.fi/pohjola/corporate-customers/payment-services-and-cash-management/bank-connection-methods/web-services?id=323430&kielikoodi=en>
- http://www.w3schools.com/xml/xml_soap.asp

RESTful Web Services

Representational State Transfer (REST)

- **Architectural pattern for web-service design**
 - **NOT a protocol (such as SOAP)**
- **Set of simple guidelines on designing WS API**
- **Goal: create APIs that are easy to use**

REST

- **Resource-oriented**
 - **Endpoints like: /posts, /users, /categories etc.**
- **Explicit use of HTTP methods**
 - **GET, PUT, DELETE etc.**
- **Stateless**

REST - Resource

- **Identified by URI (commonly a link on the Web)**

e.g. **<http://www.blogysek.com/authors>**

- “authors” resource

- **Directory-like structure**

<http://www.blogysek.com/authors> – all authors

<http://www.blogysek.com/authors/{id}> – single author

<http://www.blogysek.com/authors/{id}/articles> – all articles belonging to the author

REST - Operations

- **Use of HTTP Methods**
- **GET - read data**

```
GET /posts HTTP/1.1
```

```
//response
```

```
HTTP/1.1 200 OK
```

```
{  
  "id": 4,  
  "content": "Good morning, we've talked the whole night  
              through.",  
  "createdBy": "Duelling Cavalier"  
}
```

REST - Operations

- **Use of HTTP Methods**
- **GET - query data**

```
GET /posts?author="Duelling Cavalier" HTTP/1.1
```

```
//response
```

```
HTTP/1.1 200 OK
```

```
{  
  "id": 4,  
  "content": "Good morning, we've talked the whole night  
              through.",  
  "createdBy": "Duelling Cavalier"  
}
```

REST - Operations

- **Use of HTTP Methods**
- **DELETE - remove data**

```
DELETE /posts/12 HTTP/1.1
```

//response

```
HTTP/1.1 204 No Content
```


REST - Operations

- **Use of HTTP Methods**
- **POST - create/update data**

```
POST /posts/ HTTP/1.1
```

```
{  
  "content": "Good morning, we've talked the whole night  
             through.",  
  "createdBy": "Duelling Cavalier"  
}
```

//response

```
HTTP/1.1 201 Created
```

```
Location: /posts/63636
```

REST - Operations

- **Use of HTTP Methods**
- **PUT - create/update data**

```
PUT /posts/12 HTTP/1.1
```

```
{  
  "content": "Good morning, we've talked the whole night  
             through.",  
  "createdBy": "Duelling Cavalier"  
}
```

//response

```
HTTP/1.1 201 Created
```

```
Location: /posts/12
```

REST - Operations

- **Use of HTTP Methods**

- **PUT vs POST**

- **POST - when we do not know exact resource location**

POST /posts/ HTTP/1.1

- e.g. when server generates IDs for new resources

- **PUT - when we do know the exact resource location**

PUT /posts/12 HTTP/1.1

REST - Operations

- **Use of HTTP Methods**
 - **PUT vs POST**
 - **POST** - repeated sending of the same data does not guarantee the same result each time
 - e.g. multiple resources are created
 - **PUT** - repeated sending of the same data always has the same result
- **PUT is IDEMPOTENT operation**

REST - Operations

- **Use of HTTP Methods**
- **PUT vs POST**

Both can be used for creating new or updating existing resources

POST create:

`POST /posts/ HTTP/1.1`

```
{  
  "content": "Good morning, we've talked the whole night  
             through.",  
  "createdBy": "Duelling Cavalier"  
}
```

REST - Operations

- **Use of HTTP Methods**
- **PUT vs POST**

Both can be used for creating new or updating existing resources

POST update:

POST /posts/ HTTP/1.1

```
{  
  "id": "12",  
  "content": "Good morning, we've talked the whole night  
              through.",  
  "createdBy": "Duelling Cavalier"  
}
```

REST - Operations

- **Use of HTTP Methods**
- **PUT vs POST**

Both can be used for creating new or updating existing resources

PUT create (presuming post 12 doesn't exist):

```
PUT /posts/12 HTTP/1.1
```

```
{  
  "content": "Good morning, we've talked the whole night  
             through.",  
  "createdBy": "Duelling Cavalier"  
}
```

REST - Operations

- **Use of HTTP Methods**
- **PUT vs POST**

Both can be used for creating new or updating existing resources

PUT update (presuming post 12 exists):

```
PUT /posts/12 HTTP/1.1
```

```
{  
  "content": "Good morning, we've talked the whole night  
             through.",  
  "createdBy": "Duelling Cavalier"  
}
```


REST - Operations

- **Use of HTTP Methods**
- **OPTIONS - list of allowed methods**

```
OPTIONS /posts/ HTTP/1.1
```

```
//response
```

```
HTTP/1.1 200 OK
```

```
Allow: HEAD,GET,DELETE,OPTIONS
```

REST - Operations

- **Use of HTTP Methods**
- **PATCH - partial modification of a resource**

```
PATCH /posts/12 HTTP/1.1  
  
{  
  "createdBy": "Gene Kelly"  
}
```

//response

```
HTTP/1.1 200 OK
```

REST - Operations

- **Design notes:**
 - **You do not have to implement all methods for all resources**
 - **Which methods you allow for a resource is up to you**
 - **What if I really need method name in URI?**
 - **e.g. advanced search (too many parameters for URI query parameters)**

REST - Operations

- **Design notes:**

- **What if I really need method name in URI?**

- **e.g. advanced search (too many parameters for URI query parameters)**

`POST /posts/queries HTTP/1.1`

or

`POST /posts/forms/search HTTP/1.1`

REST – Data Format

- **Any standard data serialization format that suits you (or support multiple)**
 - **XML**
 - **JSON**
 - **JSON takes less space than XML :)**
 - **YAML**

REST - HATEOAS

- **Hypertext As The Engine Of Application State**

Use hypertext to allow clients to traverse your API:

```
GET /posts/4 HTTP/1.1
```

```
//response
```

```
HTTP/1.1 200 OK
```

```
<post id="4">
```

```
  <content>
```

```
    Good morning, we've talked the whole night through.
```

```
  </content>
```

```
  <createdBy>Duelling Cavalier</createdBy>
```

```
  <link rel="author" href="/authors/22"/>
```

```
</post>
```

REST - Summary

- **Directory-like, resource oriented endpoint structure**
- **HTTP methods for operations**
 - **all APIs look similar**
- **You do not need to follow all the guidelines for your API to be usable**

REST - Sources

- [**http://restcookbook.com**](http://restcookbook.com)
- [**http://www.restapitutorial.com**](http://www.restapitutorial.com)
- [**http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html**](http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html)
- [**https://apiary.io/**](https://apiary.io/)

WebSockets

Motivation

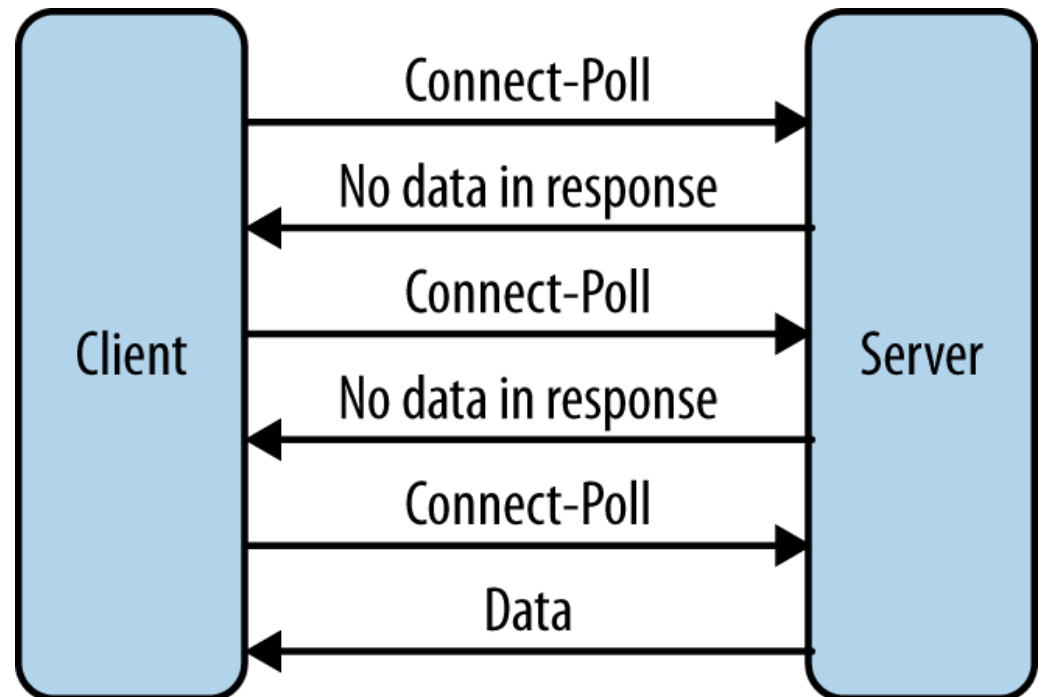
- **Some applications need to notify users of new events “real-time”**
 - e.g. an auction system
- **But standard scenario on the web is that server sends information to clients only “on request”**

Polling

Clients ask server periodically for new information (“Are we there, yet?”)

If there is none,
server response is empty

Load issues on the server
side



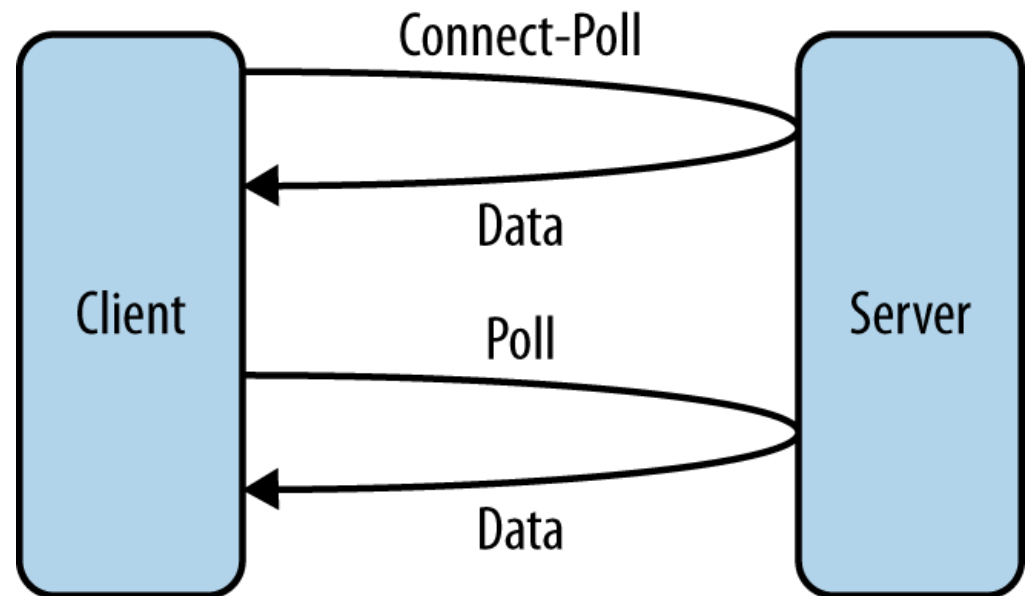
Img src: http://enterprisewebbook.com/ch8_websockets.html

Long-Polling

Clients still ask server periodically for new information

Server does not answer until there are some data to return

Or until specified time passes



Img src: http://enterprisewebbook.com/ch8_websockets.html

WebSockets

- **Messaging protocol for two-way communication**
 - Messages can be sent simultaneously in both directions
- **TCP-based**
- **Supported by most modern browsers (HTML 5)**

WebSockets

- **Connection established using HTTP handshake**
- **Use the same ports as HTTP (80 and 443)**
- **ws:// and wss:// URI schemes**
 - `ws://www.myserver.com/chat`

WebSockets - handshake

- **Connection established using HTTP handshake**

Client sends:

GET /chat HTTP/1.1

Host: www.myserver.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://www.myserver.com

WebSockets - handshake

- **Connection established using HTTP handshake**

If server supports websocket, the response is:

```
HTTP/1.1 101 Switching Protocols
```

```
Upgrade: websocket
```

```
Connection: Upgrade
```

```
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

```
Sec-WebSocket-Protocol: chat
```


WebSockets - handshake

- **Connection established using HTTP handshake**
- **After a successful handshake**
 - Protocol is switched from HTTP to WebSocket (bi-directional)
 - Uses the same TCP connection

WebSockets - Message

- **Each message is split into either one or multiple data frames**

```

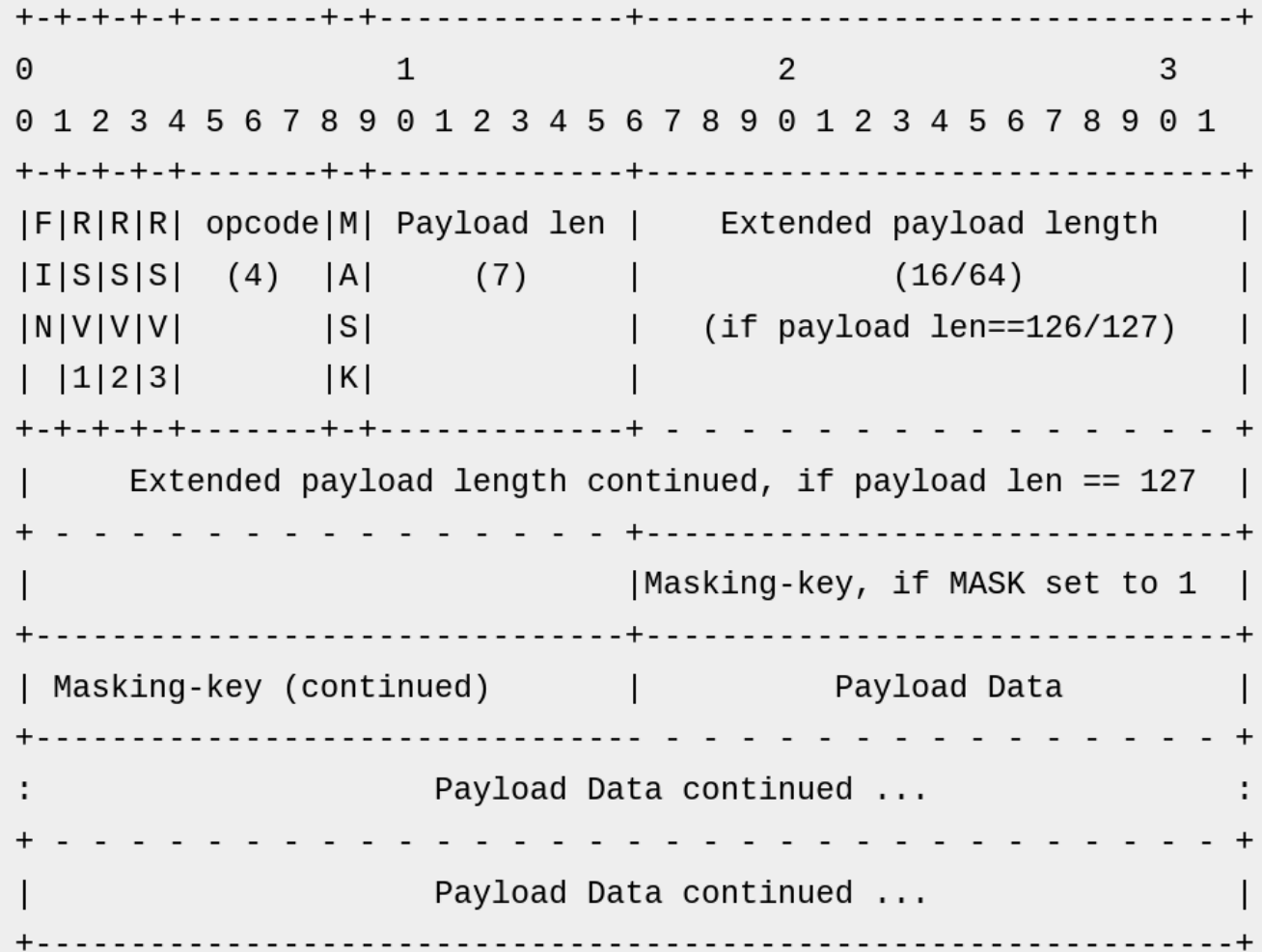
+--+--+--+--+-----+--+--+--+--+-----+--+--+--+--+-----+--+--+--+--+-----+
0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+-----+--+--+--+--+-----+--+--+--+--+-----+--+--+--+--+-----+
|F|R|R|R| opcode|M| Payload len |      Extended payload length      |
|I|S|S|S|  (4)  |A|      (7)      |      (16/64)                      |
|N|V|V|V|      |S|                  | (if payload len==126/127)      |
| |1|2|3|      |K|                  |                                  |
+--+--+--+--+-----+--+--+--+--+-----+ - - - - - - - - - - - - - - - - - +
|      Extended payload length continued, if payload len == 127      |
+ - - - - - - - - - - - - - - - - - - +-----+-----+-----+-----+
|                                  |Masking-key, if MASK set to 1      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Masking-key (continued)         |      Payload Data                  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
:                               Payload Data continued ...              :
+ - - - - - - - - - - - - - - - - - - +-----+-----+-----+-----+
|                               Payload Data continued ...              |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

WebSockets - Message

- **FIN (1 bit)**

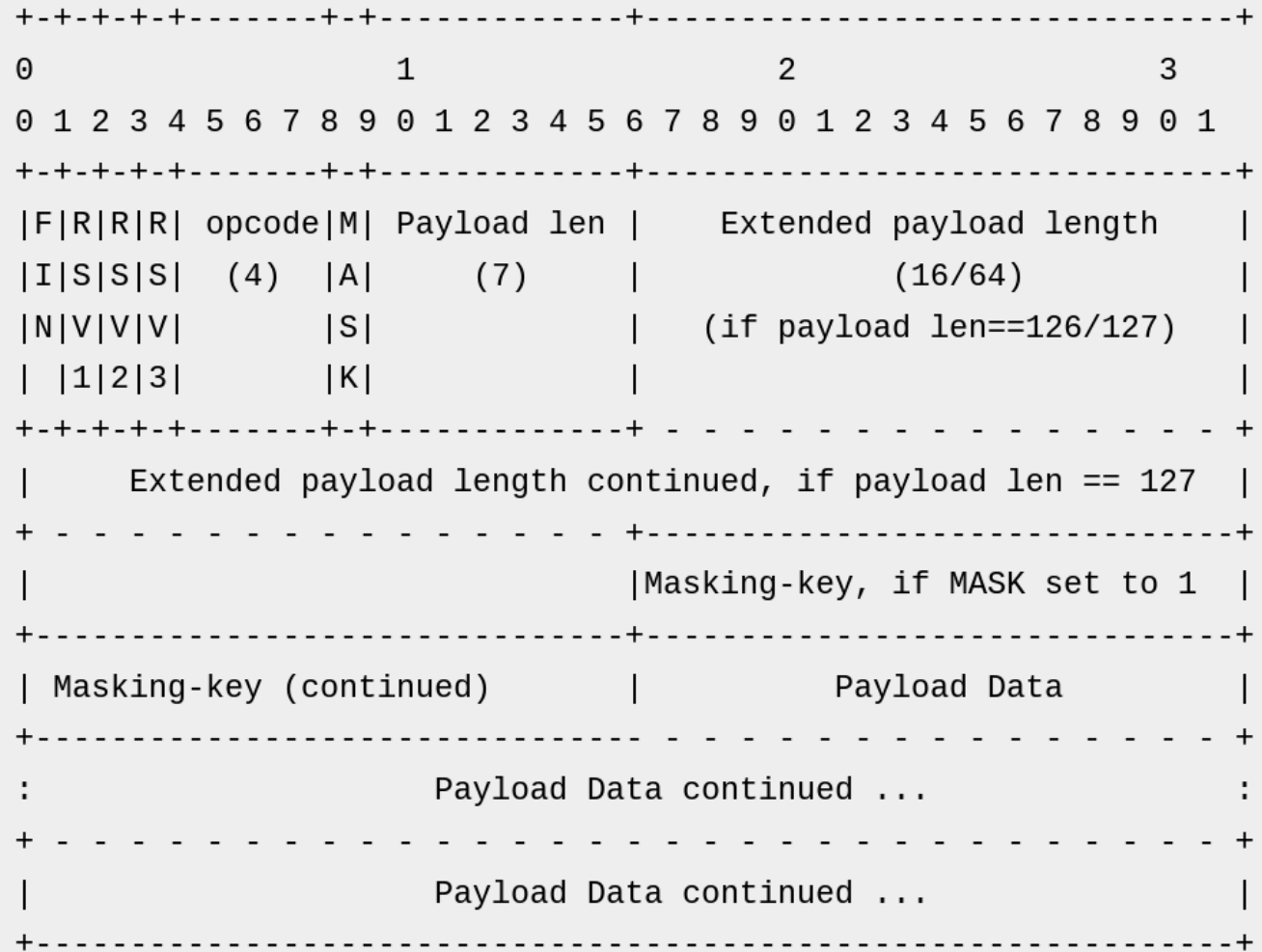
- Indicates whether this is the last frame of the message



WebSockets - Message

- **RSVx (1 bit)**

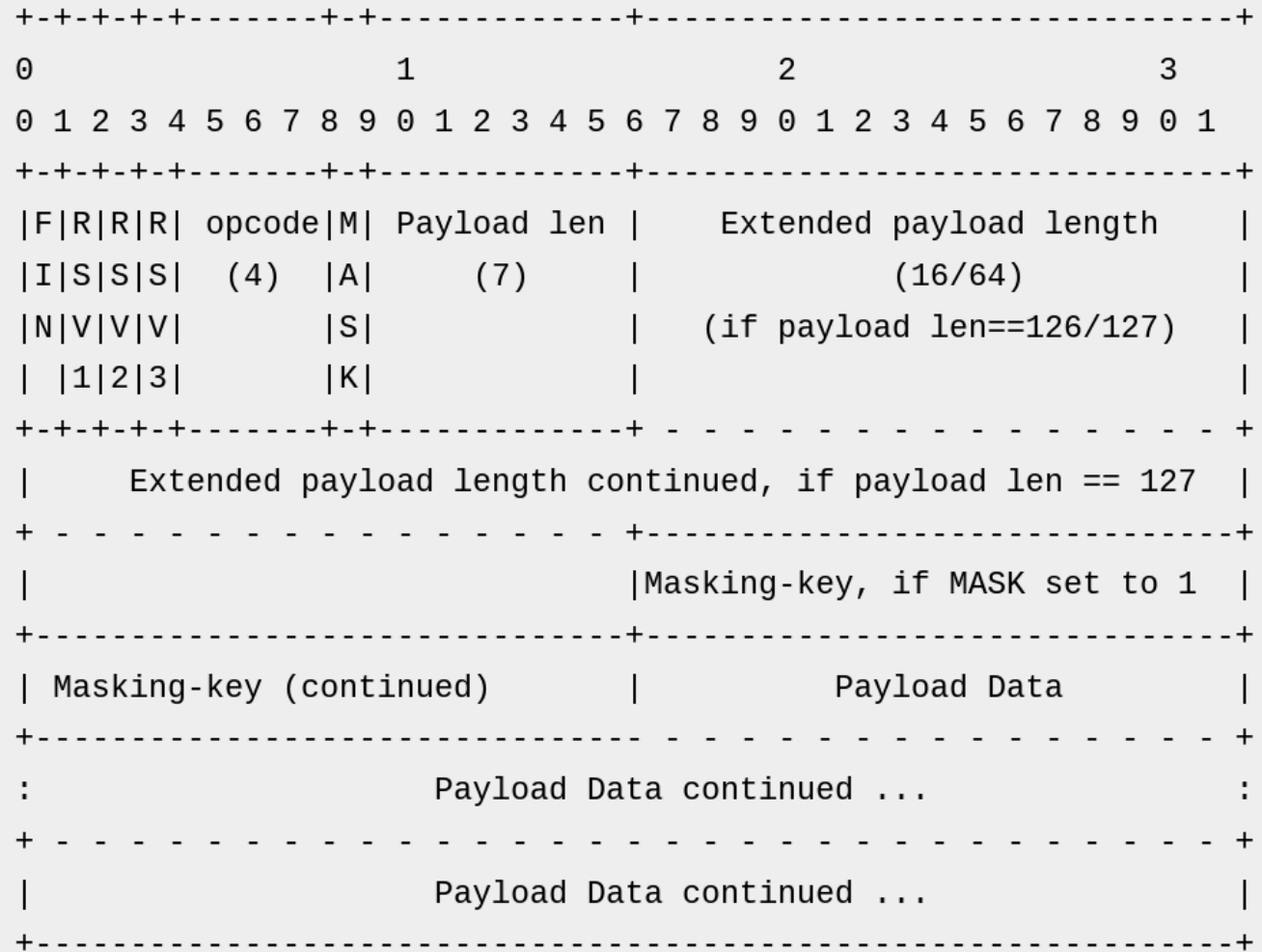
- Reserved bits for future protocol extensions
- always zero for now



WebSockets - Message

- **opcode (4 bits)**

- Frame type
- 0x01 – UTF-8 data
- 0x02 – binary data
- 0x00 – continued payload
- ...



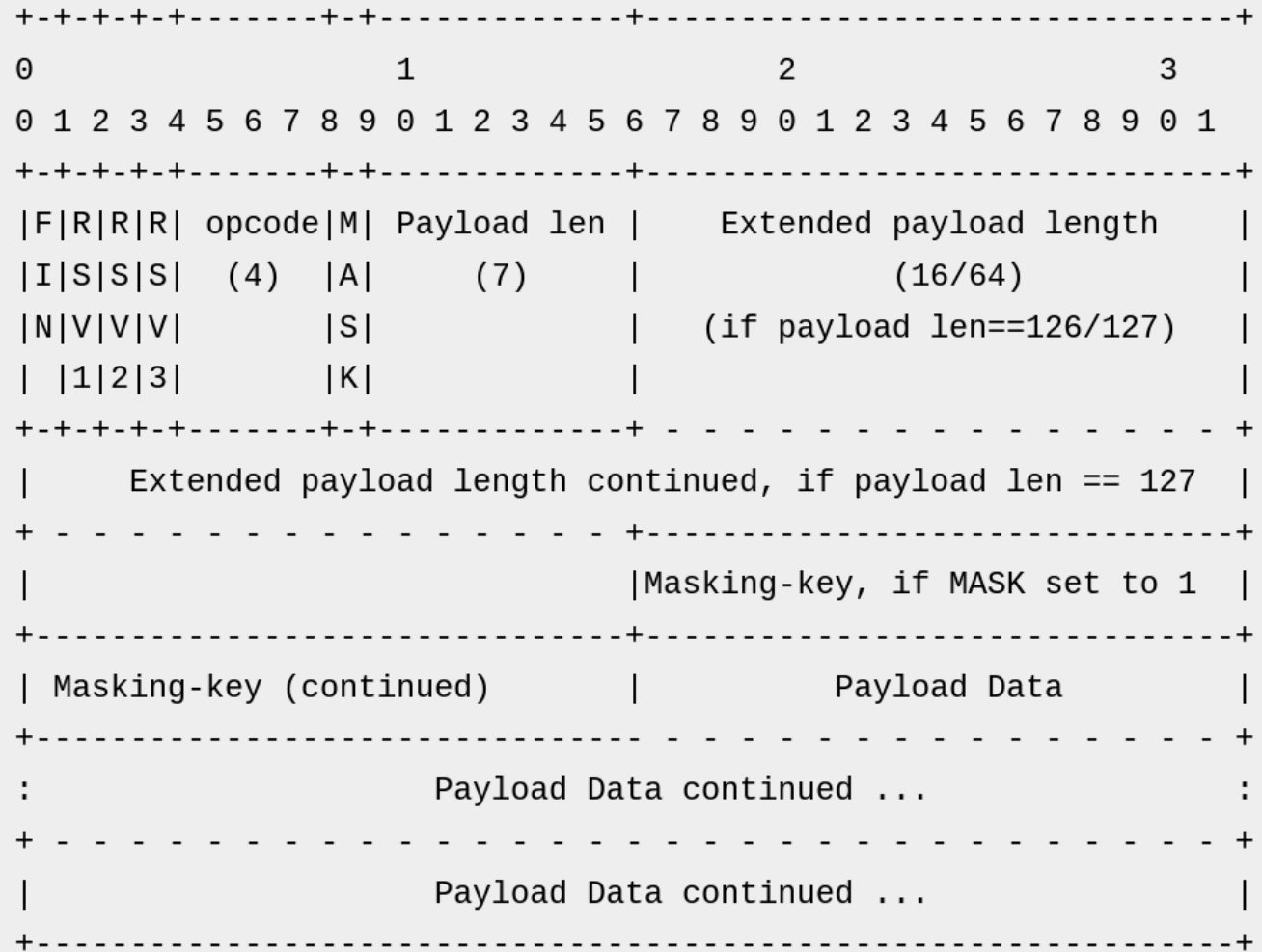
WebSockets - Message

- **mask (1 bit)**

- True if frame is masked

- **Masking-key**

- If mask == 1
- Used to XOR the payload



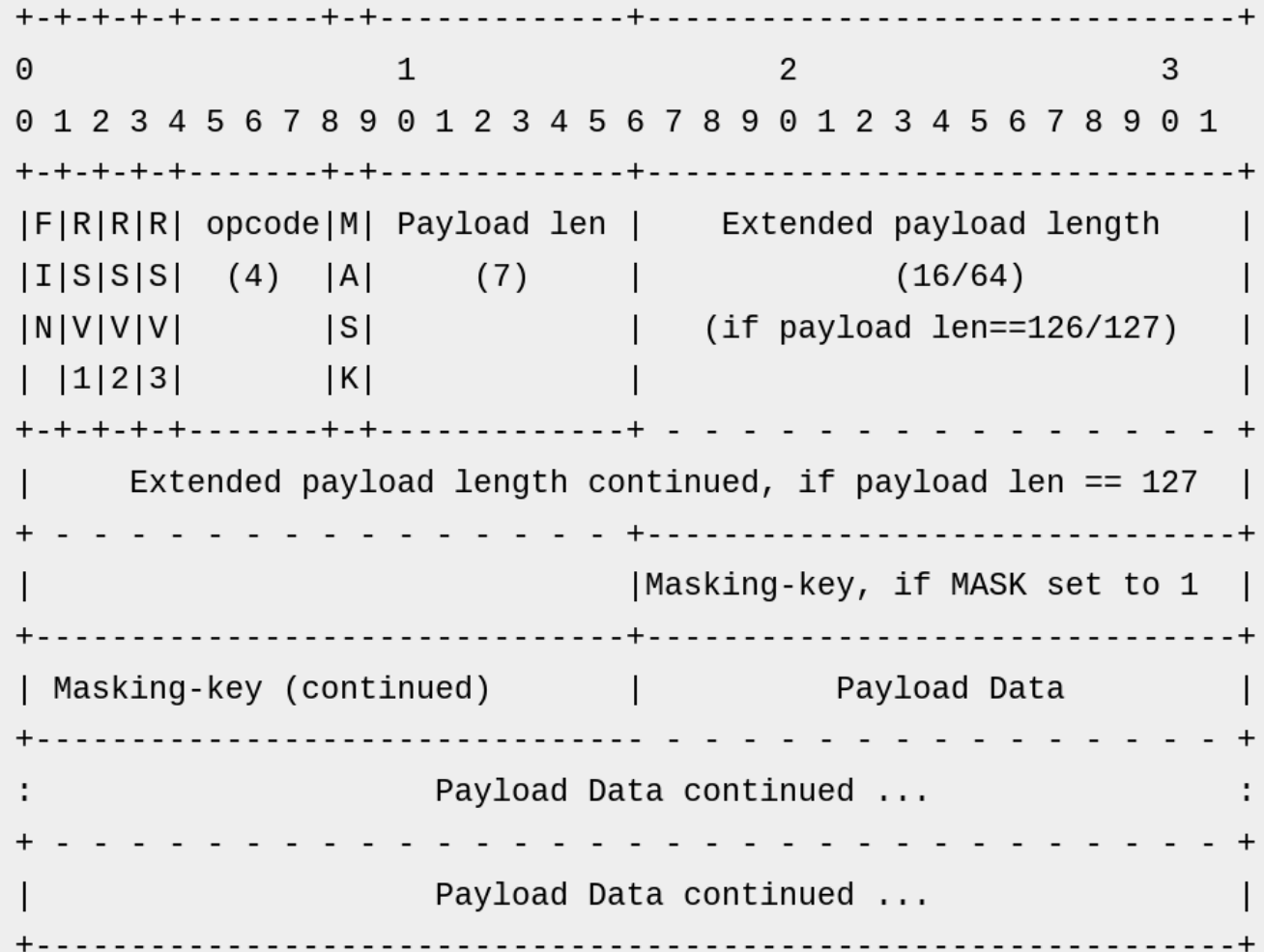
WebSockets - Message

• Payload length

- 0-125 indicate the length of the payload.
- 126 means that the following 2 bytes indicate the length.
- 127 means the next 8 bytes indicate the length.

• Payload data

- The actual sent data



WebSockets - Message

- **First frame states message data type using opcode 0x01 or 0x02**
- **The following frames have opcode 0x00**
- **Last frame has FIN == 1**

WebSockets - Heartbeat

- **Both sides need to be aware if the counterpart is still there**
 - Ideally before the next “send” attempt
 - When asking, “ping” message is sent – opcode 0x9
 - May contain application data (up to 125 bytes)
 - As an answer, “pong” message is sent – opcode 0xA
 - Must contain the same application data as the “ping” message

WebSockets - Closing Connection

- **Both sides may close the connection**
- **Opcode 0x08**
- **Additional data such as exit code, reason (arbitrary string) or whether the closing was “clean” can be sent**

WebSockets - HTML 5 API

```
//attempt connection  
  
var socket = new WebSocket("ws://myserver.com/chat");  
  
//handler called on successful connection  
socket.onopen = function() {  
    console.log("Connection successful");  
}  
  
//handler called when a message is received  
socket.onmessage = function(msg) {console.log(msg.data);}  
  
//sending a message  
socket.send("Ahoj svete!");
```

Sources

- <https://tools.ietf.org/html/rfc6455#page-27>
- http://enterprisewebbook.com/ch8_websockets.html
- <https://www.websocket.org/aboutwebsocket.html>
- <http://lucumr.pocoo.org/2012/9/24/websockets-101/>
- <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html>

Thank You!