Lecture 4
summary of Java SE & Network Programming – section 2

presentation
DAD – Distributed Applications Development
Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro
Cristian Toma – Business Card

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00
Agenda for Lecture 4

1. Recap Networking
2. UDP & TCP Network Programming
3. Exchange Ideas
Networking Recapitulation

Networking IP, UDP and TCP programming, TCP/IP state machine, SNMP, SMTP
1. Networking TCP/IP Stack

HOW TCP/IP Works:

http://buildingautomationmonthly.com/tcpip-an-overview/

http://www.barrgroup.com/Embedded-Systems/How-To/Embedded-TCP-IP
1. Networking TCP/IP Stack

TCP/IP Stack Model:

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Network Layer
- Data-Link Layer
- Physical Layer

TCP/IP Protocol Suite:

- Internet Protocol (IP/IPv4, IPv6)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Internet Control Message Protocol (ICMP)
- Address Resolution Protocol (ARP)
- Reverse Address Resolution Protocol (RARP)
- IP Support Protocols
- IP Routing Protocols

Protocols:
- Telnet
- FTP
- SMTP
- DNS
- RIP
- SNMP
- HTTP
- Telnet
- "r" Commands
- IRC
- DNS
- BOOTP
- SNMP
- FTP
- FTP
- HTTP
- Gopher
- Telnet
- "r" Commands
- IRC

Links:
- http://talktoanit.com/c/?p=78
## 1. Networking TCP/IP Stack

### ISO/OSI Model vs. TCP/IP:

<table>
<thead>
<tr>
<th>ISO/OSI Model</th>
<th>TCP/IP DoD Model</th>
<th>OSI Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Layer (Services Layers 5,6,7)</td>
<td>HTTP: port 80, HTTPS/TLS/SSL: port 443</td>
<td>Application Layer (7)</td>
</tr>
<tr>
<td></td>
<td>DNS: port 53, TFTP: port 69, DHCP/BootP: port 67,68, SNMP: port 162,161, NTP: port 123, Syslog: port 514</td>
<td>Serves the King/User</td>
</tr>
<tr>
<td>Transport Layer (Host to Host Layer 4)</td>
<td>TCP: protocol 6, UDP: protocol 17</td>
<td>Presentation Layer (6)</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>Translator. Reformats, encrypts/de-crypts, compress/de-compress</td>
</tr>
<tr>
<td>Internet Layer (Network Layer 3)</td>
<td></td>
<td>Session Layer (5)</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>Negotiator. Establishes, manages and ends sessions.</td>
</tr>
<tr>
<td>Network Access Layer 1 &amp; 2</td>
<td>Ethernet, PPP, Frame Relay, MAC addresses, ARP</td>
<td>Transport Layer (4)</td>
</tr>
<tr>
<td>PDU: Frame</td>
<td>Ethernet, PPP, Frame Relay, MAC addresses, ARP</td>
<td>Middle Manager. Segment ID/Assembly</td>
</tr>
<tr>
<td>Network Access Layer 1 &amp; 2</td>
<td>Electrons, RF or Light</td>
<td>Network Layer (3)</td>
</tr>
<tr>
<td>PDU: Bits or Data Stream</td>
<td>Electrons, RF or Light</td>
<td>Mail Room Guy. IP Addressing/Routing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data-Link Layer (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Envelope Stuffer. Organizes bits into frames</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Physical Layer (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Truck. Movement of bits</td>
</tr>
</tbody>
</table>

http://buildingautomationmonthly.com/tcpip-an-overview/
1. Networking  TCP/IP Stack

TCP/IP Message Encapsulation:

http://en.wikipedia.org/wiki/Internet_protocol_suite
1. Networking TCP/IP Stack

TCP/IP Message Flow:

http://buildingautomationmonthly.com/tcpip-an-overview/
1. Networking TCP/IP Stack

1. Networking TCP/IP Stack

TCP/IP App/Port Multiplexing:
1. Networking  TCP/IP Stack

What is running on port 80?

Applications Have Changed – Firewalls Have Not

- The gateway at the trust border is the right place to enforce policy control
  - Sees all traffic
  - Defines trust boundary

- BUT...Applications Have Changed
  - Ports ≠ Applications
  - IP Addresses ≠ Users
  - Packets ≠ Content

Need to Restore Visibility and Control in the Firewall
1. Networking TCP/IP Stack

IP Header - RFC 791 – Submasking + Routing + NAT:
1. Networking TCP/IP Stack

UDP Header – RFC 768 – Connection-less vs. TCP Header – RFC 793 – Connection-oriented
1. Networking TCP/IP Stack

TCP State Machine – RFC 793:

- **CLOSED**
  - **CONNECTION**: SYN, SYN+ACK (Step 1 of the 3-way-handshake)
  - **CLOSE**: CLOSE

- **LISTEN**
  - **SYNC**: SYN (Step 2 of the 3-way-handshake)
  - **RST**: RST
  - **SEND**: SEND/SYN
  - **SYN**: SYN
  - **SYN+ACK**: SYN+ACK

- **ESTABLISHED**
  - **DATA EXCHANGE**: ACK
  - **CLOSE/FIN**: CLOSE/FIN
  - **FIN/ACK**: FIN/ACK

- **FIN_WAIT_1**
  - **FIN**: FIN
  - **ACK**: ACK

- **FIN_WAIT_2**
  - **FIN**: FIN
  - **ACK**: ACK

- **CLOSING**
  - **FIN**: FIN
  - **ACK**: ACK

- **TIME_WAIT**
  - **TIMEOUT**: Timeout

- **LAST_ACK**
  - **CLOSE/FIN**: CLOSE/FIN

- **CLOSED**
  - **(Go back to start)**

- **ACTIVE OPEN**
  - **CLOSE_WAIT**
  - **FIN/ACK**: FIN/ACK

- **PASSIVE OPEN**
  - **LAST_ACK**
  - **FIN/ACK**: FIN/ACK
1. Networking TCP/IP Stack

TCP/IP NAT & PAT:

1. Networking TCP/IP Stack

TCP/IP NAT:

An IP address is either local or global
Local IP addresses are seen in the inside network
Global IP addresses are seen in the Outside network

TCP/IP NAT:

- **My Network**
  - **SA**: 10.6.1.20
  - **DA**: 10.6.1.20

- **Internet**
  - **SA**: 171.69.68.10
  - **DA**: 171.69.68.10

### NAT Terminology « Inside Addressing »

<table>
<thead>
<tr>
<th>Pro</th>
<th>Inside Global</th>
<th>Inside Local</th>
<th>Outside Local</th>
<th>Outside Global</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>171.69.68.10</td>
<td>10.6.1.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10.6.1.20 is inside local address
171.69.68.10 is inside global address

1. Networking TCP/IP Stack

TCP/IP NAT:

NAT Terminology "Outside Addressing"

- **Pro**: --
- **Inside Global**: --
- **Inside Local**: 192.168.1.20
- **Outside Local**: 192.168.1.20
- **Outside Global**: 171.69.68.10

192.168.1.20 is inside local address
171.69.68.10 is inside global address

1. Networking TCP/IP Stack

TCP/IP PAT:

**Basic Concepts of PAT**

**My Network**

10.6.1.2

10.6.1.6

SA 10.6.1.2:2031

SA 10.6.1.6:1506

**Internet**

171.69.68.10:2031

171.69.68.10:1506

**Internet/Intranet**

Port Address Translation (PAT) extends NAT from "1 to 1" to "many-to-1" by associating the source port with each flow.

**Figure 5**

Unique Source Port per Translation Entry

<table>
<thead>
<tr>
<th>Pro</th>
<th>Inside Global</th>
<th>Inside Local</th>
<th>Outside Local</th>
<th>Outside Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>171.69.68.5:1405</td>
<td>10.6.15.2:1405</td>
<td>204.71.200.69:80</td>
<td>204.71.200.69:80</td>
</tr>
</tbody>
</table>

PAT (Port Address Translation) includes ports in addition to IP addresses:
- Many-to-one translation
- Maps multiple IP addresses to 1 or a few IP addresses
- Unique source port number identifies each session
- Conserves registered IP addresses
- Also called NAPT in IETF documents
1.2 TCP/IP Networking Programming

TCP Handshake:

![Diagram of TCP Handshake](http://www.cisco.com/web/about/ac123/ac147/ac174/ac196/about_cisco_ipj_archive_article09186a00800c8417.html)
Section Conclusion

Fact: **DAD needs Networking**

In few samples it is simple to remember: UDP and TCP programming is useful for HTC ... didactical samples: SNMP over UDP for monitoring, SMTP/IMAP4/POP3 over TCP for e-mail notification, FTP over TCP for file and data transfer, HTTP over TCP for web & web services, and in general TCP for RPC/RMI, CORBA, JMS, SOAP, P2P-JXTA – for distributed computing and systems.
UDP Client-Server programming, SNMP client sample, TCP Client-Server programming, SMTP client sample

**Network** UDP & TCP Programming
2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

UDP Server:
- socket()
- bind()
- recvfrom()
- sendto()
- close()

Network:

UDP Client:
- socket()
- sento()
- recvfrom()
- close()
2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

```java
package eu.ase.net udp;

import java.io.*;
import java.net.*;

public class UDPServer {
    public static void main(String[] args) {
        // get a datagram socket
        DatagramSocket socket = null;
        byte[] bufResp = null;
        byte[] bufRecv = null;
        try {
            socket = new DatagramSocket(778); // it is correct because this constructor executes "bind"
            while(true) {
                bufRecv = new byte[256];
                // receive request
                DatagramPacket packet = new DatagramPacket(bufRecv, bufRecv.length);
                socket.receive(packet);

                // figure out response
                String respString = new String("OK");
                bufResp = respString.getBytes();

                // send the response to the client at "address" and "port"
                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                packet = new DatagramPacket(bufResp, bufResp.length, address, port);
                socket.send(packet);
            }
        }
    }
}
```
2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

```java
package eu.ase.net.udp;
import java.io.*;
import java.net.*;

public class UDPClient {
    public static void main(String[] args) throws IOException {
        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();

        // send request
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName("127.0.0.1");
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 778);
        socket.send(packet);

        // get response
        byte[] bufResp = new byte[256];
        packet = new DatagramPacket(bufResp, bufResp.length);
        socket.receive(packet);

        // display response
        String received = new String(packet.getData());
        System.out.println("Client de la server: ", received);

        // close socket
        socket.close();
    }
}
```
2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:

TCP Server
- socket()
- bind()
- listen()
- accept()
- read()/recv()
- write()/send()
- close()

Network
- Establishing the connection / TCP Handshake

TCP Client
- socket()
- connect()
- write()/send()
- read()/recv()
- close()
2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:

```
ServerSocket serverSocket = null;
Socket clientSocket = null;
boolean listening = true;
OutputStream os = null; PrintWriter out = null;
InputStream is = null; BufferedReader in = null;
String inputLine = null, outputLine = null;

//SEVERSOCKET = SOCKET+BIND+LISTEN
serverSocket = new ServerSocket(4801);
//ACCEPT

//STABILIREA CONEXIUNII
//CONNECT = OUT2SERVER + INfromSERVER

is = clientSocket.getInputStream();
in = new BufferedReader(new InputStreamReader(is));

os = clientSocket.getOutputStream();
out = new PrintWriter(os, true);

while ((inputLine = in.readLine()) != null) {   
    System.out.println(inputLine);
    outputLine = new String("OK");
    out.println(outputLine);
    out.flush();
    if (outputLine.compareTo("La revedere!") == 0) {break;} 
}
```

```
Socket clientSocket = null;
PrintWriter outC = null;
BufferedReader inC = null;
clientSocket = new Socket(args[0],
Integer.parseInt(args[1]));//SOCKET

//STABILIREA CONEXIUNII
//OUT2SERVER
outC = new PrintWriter(clientSocket.getOutputStream(), true);

//INfromSERVER
inC = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

String lin = "";
outC.println("As vrea sa ma conectez.");//SEND
lin = inC.readLine(); //RECV
System.out.println("Server: " + lin);
```

```
2.3 SNMP – RFC 1157

What is the necessity for SNMP – Simple? Network Management Protocol?
2.3 SNMP — RFC 1157

ASN.1 DER and BER:

RFC1157-SNMP DEFINITIONS ::= BEGIN

IMPORTS
ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
FROM RFC1155-SMI;

-- top-level message
Message ::= SEQUENCE {
  version -- version-1 for this RFC
    INTEGER { version-1(0) }
  },

  community -- community name
    OCTET STRING,

  data -- e.g., PDUs if trivial
    ANY -- authentication is being used
}

-- protocol data units
PDUs ::= CHOICE {
  get-request
    GetRequest-PDU,

  get-next-request
    GetNextRequest-PDU,

  get-response
    GetResponse-PDU,

  set-request
    SetRequest-PDU,

  trap
    Trap-PDU
}

-- PDUs
GetRequest-PDU ::= [0] IMPLICIT PDU

GetNextRequest-PDU ::= [1] IMPLICIT PDU

GetResponse-PDU ::= [2] IMPLICIT PDU

SetRequest-PDU ::= [3] IMPLICIT PDU

PDU ::= SEQUENCE {
  request-id
    INTEGER,

  error-status -- sometimes ignored
    INTEGER {
      noError(0),
      tooBig(1),
      noSuchName(2),
      badValue(3),
    }
}

### 2.3 SNMP – RFC 1157

**ASN.1 DER and BER:**

Some of the **data types** used in ASN.1 are presented in the following listing:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sort</th>
<th>Tag Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Primitive</td>
<td>0x01</td>
<td>Boolean value: yes/no</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Primitive</td>
<td>0x02</td>
<td>Negative and positive integers</td>
</tr>
<tr>
<td>BIT STRING</td>
<td>Primitive</td>
<td>0x03</td>
<td>Bit sequence</td>
</tr>
<tr>
<td>OCTET STRING</td>
<td>Primitive</td>
<td>0x04</td>
<td>Byte sequence (1 byte = 8 bits = 1 octet)</td>
</tr>
<tr>
<td>NULL</td>
<td>Primitive</td>
<td>0x05</td>
<td>A null value</td>
</tr>
<tr>
<td>OBJECT IDENTIFIER</td>
<td>Primitive</td>
<td>0x06</td>
<td>An object identifier, which is a sequence of integer components that identify an object such as an algorithm or attribute type</td>
</tr>
<tr>
<td>PrintableString</td>
<td>Primitive</td>
<td>0x13</td>
<td>An arbitrary string of printable characters</td>
</tr>
<tr>
<td>T61String</td>
<td>Primitive</td>
<td>0x14</td>
<td>An arbitrary string of T.61 (eight-bit) characters</td>
</tr>
<tr>
<td>IA5String</td>
<td>Primitive</td>
<td>0x16</td>
<td>An arbitrary string of IA5 (ASCII) characters</td>
</tr>
<tr>
<td>UTCTime</td>
<td>Primitive</td>
<td>0x17</td>
<td>A &quot;coordinated universal time&quot; or Greenwich Mean Time (GMT) value</td>
</tr>
</tbody>
</table>
### 2.3 SNMP – RFC 1157

**ASN.1 DER and BER:**

Some of *structured types* defined in ASN.1 are presented in the following listing:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sort</th>
<th>Tag Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE</td>
<td>Constructed</td>
<td>0x30</td>
<td>An ordered collection of one or more types</td>
</tr>
<tr>
<td>SEQUENCE OF</td>
<td>Constructed</td>
<td>0x10</td>
<td>An ordered collection of zero or more occurrences of a given type</td>
</tr>
<tr>
<td>SET</td>
<td>Constructed</td>
<td>0x31</td>
<td>An unordered collection of one or more types</td>
</tr>
<tr>
<td>SET OF</td>
<td>Constructed</td>
<td>0x11</td>
<td>An unordered collection of zero or more occurrences of a given type</td>
</tr>
<tr>
<td>A0, A1, ...</td>
<td>Constructed</td>
<td>0xAz</td>
<td>Where z = 0..F in hex and it represents the z-th element in SEQUENCE data type</td>
</tr>
</tbody>
</table>
2.3 SNMP – RFC 1157

ASN.1 TLV:

```
<table>
<thead>
<tr>
<th>T - tag</th>
<th>L - length</th>
<th>V - value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2 bytes</td>
<td>1-3 bytes</td>
<td>n bytes</td>
</tr>
</tbody>
</table>
```

TLV Object

<table>
<thead>
<tr>
<th>Tag</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x85</td>
<td>0x05</td>
<td>0x46 0x72 0x61 0x6E 0x6B</td>
</tr>
</tbody>
</table>

Tag for first names, Length of the first name, First name “Frank” in ASCII

Tag for constructed data object, Length of the constructed data object, Value of the constructed data object

```
Tc  Lc  Vc
T_{p1} L_{p1} V_{p1} T_{p2} L_{p2} V_{p2} T_{p3} L_{p3} V_{p3}
```
2.3 SNMP – RFC 1157

ASN.1 DER and BER:

<table>
<thead>
<tr>
<th>T - tag</th>
<th>L - length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b7 b6 b5 b4 b3 b2 b1 b0</td>
<td>Meaning</td>
</tr>
<tr>
<td>0 0 ... ... ... ... ...</td>
<td>Universal class</td>
</tr>
<tr>
<td>0 1 ... ... ... ... ...</td>
<td>Application class</td>
</tr>
<tr>
<td>1 0 ... ... ... ... ...</td>
<td>Context-specific class</td>
</tr>
<tr>
<td>1 1 ... ... ... ... ...</td>
<td>Private class</td>
</tr>
<tr>
<td>... ... 0 ... ... ... ...</td>
<td>Primitive data object</td>
</tr>
<tr>
<td>... ... 1 ... ... ... ...</td>
<td>Constructed data obj.</td>
</tr>
<tr>
<td>... ... Y Y Y Y Y</td>
<td>Tag code (0-30)</td>
</tr>
<tr>
<td>... ... 1 1 1 1 1</td>
<td>Pointer to the following byte (byte 2), which specifies the tag code</td>
</tr>
</tbody>
</table>

Byte 1

Byte 2

Byte 3, 4, 5
2.3 SNMP – RFC 1157

ASN.1 OID:

http://www.oid-info.com/cgi-bin/display
2.3 SNMP – RFC 1157

ASN.1 OID:

1.2.840.113549.1.1.5
({iso(1) member-body(2) us(840) rsadsi(113549)
pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)})

Sample of OID is 1.2.840.113549.1.1.5 ({iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}) for
signature obtained from SHA-1 digest on the message and RSA algorithm
applied to the digest. This encoding will be at byte level: 0x06 0x09 0x2A 0x86
0x48 0x86 0xF7 0x0D 0x01 0x01 0x05. The first byte shows that here there is a
OID – OBJECT IDENTIFIER field. The OID has 9 bytes length because of
second byte in array. Because the first bit in length field is not set the length
field has only one byte. The length field has value 9 which means the OID
structure has the payload data in 9 bytes.
According to BER, the first two numbers of any OID \((x.y)\) are encoded as one value using the formula \((40^\cdot x)+y\). The first two numbers in an OID are here 1.2. Therefore, the first two numbers of an OID are encoded as 42 or 0x2A, because \((40^\cdot 1)+2 = 42\). After the first two numbers are encoded, the subsequent numbers in the OID are each encoded as a byte. However, a special rule is required for large numbers because one byte (eight bits) can only represent a number from 0-255. This is the case for 840 and 113549. For 840 from the OID, the first bit of the first byte should be set. The number occupies enough number of bytes till the last byte of representation is not having the first bit set. In 840 case 0x48 has the first bit NOT set. So, the formula is the last hex digit from the first byte multiplied with \(2^7 = 128\) (1 bit is for multiple bytes representation) and added with the value from the second byte as long as the second byte has a value less than 0x80. In 840 case, the formula is: \(0x06\cdot 128 + 0x48\) (form 0x86 0x48) = 768 + 72 = 840. In case of 113549 we choose the bytes 0x86 0xF7 0x0D because 0x0D is the last byte with first bit (sign bit) not set. The formula for 113549 is: \(0x06\cdot 2^14 + 0x77\cdot 2^7 + 0x0d\cdot 2^0 = 6\cdot 16384 + 119\cdot 128 + 13\cdot 1 = 98304 + 15232 + 13 = 113549\). For the remaining encoding \{"pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)\}, we will have only one byte for each number: 0x01 0x01 0x05.
2.3 SNMP – RFC 1157
# 2.3 SNMP – RFC 1157

## SNMP Message (Sequence)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP message</td>
<td>A Sequence representing the entire SNMP message consisting of the SNMP version, Community String, and SNMP PDU.</td>
</tr>
<tr>
<td>SNMP Version</td>
<td>An Integer that identifies the version of SNMP. SNMPv1 = 0</td>
</tr>
<tr>
<td>SNMP Community String</td>
<td>An Octet String that may contain a string used to add security to SNMP devices.</td>
</tr>
<tr>
<td>SNMP PDU</td>
<td>An SNMP PDU contains the body of the SNMP message. There are several types of PDUs. Three common PDUs are GetRequest, GetResponse, SetRequest.</td>
</tr>
<tr>
<td>Request ID</td>
<td>An Integer that identifies a particular SNMP request. This index is echoed back in the response from the SNMP agent, allowing the SNMP manager to match an incoming response to the appropriate request.</td>
</tr>
</tbody>
</table>
An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:

- **0x00** -- No error occurred
- **0x01** -- Response message too large to transport
- **0x02** -- The name of the requested object was not found
- **0x03** -- A data type in the request did not match the data type in the SNMP agent
- **0x04** -- The SNMP manager attempted to set a read-only parameter
- **0x05** -- General Error (some error other than the ones listed above)
2.3 SNMP – RFC 1157

<table>
<thead>
<tr>
<th>Error Index</th>
<th>If an Error occurs, the Error Index holds a pointer to the Object that caused the error, otherwise the Error Index is 0x00.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Varbind List</td>
<td>A Sequence of Varbinds.</td>
</tr>
<tr>
<td>Varbind</td>
<td>A Sequence of two fields, an Object ID and the value for/from that Object ID.</td>
</tr>
<tr>
<td>Object Identifier</td>
<td>An Object Identifier that points to a particular parameter in the SNMP agent.</td>
</tr>
</tbody>
</table>

**Value**

- **SetRequest PDU** -- **Value** is applied to the specified OID of the SNMP agent.
- **GetRequest PDU** -- **Value** is a Null that acts as a placeholder for the return data.
- **GetResponse PDU** -- The returned Value from the specified OID of the SNMP agent.
2.3 SNMP – RFC 1157

![Diagram of SNMP message and PDU structure with detailed types and values, including SNMP Version, Community String, Request ID, Error, Error Index, Varbind List, and Value fields.]
D.1 A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host bar.com, to Jones, Green, and Brown at host foo.com. Here we assume that host bar.com contacts host foo.com directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host foo.com.

S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-3BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>

S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
Base64 encoding is used in practice usually for transport over the network and heterogeneous environments binary code such as pictures or executable code. The techniques is very simple: to transform each 3 bytes values into 4 bytes value in order to avoid to obtain values greater then 127 per byte.

For instance, if the scope is to encode the word “Man” into Base64 encoding then it is encoded as “TWFu”. Encoded in ASCII (in ISO 8859-1, one value per byte), M, a, n are stored as the bytes 77 (0x4D), 97 (0x61), 110 (0x6E), which are 01001101, 01100001, 01101110 in base 2.

As this example illustrates, the encoding converts 3 not encoded bytes (in this case, ASCII characters) into 4 encoded ASCII characters. If the output is not multiple of 4 bytes then the ‘=’ sign is put in order to supply the padding. Must be considered the padding with = (64 value) in case of multiple 4 bytes number.
2.4 SMTP – RFC 2821

TCP SYN

TCP SYN ACK

TCP ACK

SMTP Command TCP PUSH, ACK

SMTP Response TCP PUSH, ACK

EHLO mail.ase.ro

AUTH LOGIN
c2VjaXRj
zoGvvaXzhbiM=

MAIL FROM:<securitateinformatica@ase.ro>

RCPT TO:<cristianvtoma@gmail.com>
2.4 SMTP – RFC 2821

Date: Sun Jun 01 15:24:37 BST 2008
Subject: SECITC 2008 Account Registration
To: cristianvtoma@gmail.com
From: <secitc@ase.ro>

Thank you for registering at SECITC 2008 – I Information Technology and Communication. Please activate your account by clicking the

Best regards,
SECITC 2008 Team
2.4 SMTP – RFC 2821

Simple Mail Transfer Protocol

Response: 220 s-mail-1.ase.ro Groupwise Internet Agent 7.0.1 Copyright (C) 1993-2000 Novell, Inc. All rights reserved. Ready.

Simple Mail Transfer Protocol (smtp), 118 bytes
Section Conclusions

Java Network Programming uses for UDP: DatagramSocket and DatagramPacket classes on both server and client side.


For both server and client, it is necessary to create byte/char Input (socket.getInputStream()) and Output (socket.getOutputStream()) streams between the Random Access Memory – RAM and the network communications channel.

As practical approach, also developed in the laboratory activities, there is a need to implement “raw” SNMP request, but third party libraries – snmp4j.org or product – Nagios, Cacti, Zabbix may be used.

As practical approach, there is a need to implement “raw” SMTP in order to send e-mails, but Sun/Oracle or third party libraries may be used – java.mail.*.
Network Programming & Java Sockets

Communicate & Exchange Ideas
Questions & Answers!

But wait...

There’s More!
Thanks!

DAD – Distributed Application Development
End of Lecture 4 – summary of Java SE & Network Programming – section 2