INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Q.2111

Amendment 3
(10/2003)

SERIES Q: SWITCHING AND SIGNALLING

Broadband ISDN – Signalling ATM adaptation layer (SAAL)

B-ISDN ATM adaptation layer – Service specific connection oriented protocol in a multilink and connectionless environment (SSCOPMCE)

Amendment 3: API for SSCOPMCE over Ethernet and UDP port number

ITU-T Recommendation Q.2111 (1999) – Amendment 3

ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING

SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE	Q.1–Q.3
INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING	Q.4–Q.59
FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN	Q.60–Q.99
CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS	Q.100–Q.119
SPECIFICATIONS OF SIGNALLING SYSTEMS No. 4, 5, 6, R1 AND R2	Q.120–Q.499
DIGITAL EXCHANGES	Q.500–Q.599
INTERWORKING OF SIGNALLING SYSTEMS	Q.600–Q.699
SPECIFICATIONS OF SIGNALLING SYSTEM No. 7	Q.700–Q.799
Q3 INTERFACE	Q.800–Q.849
DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1	Q.850–Q.999
PUBLIC LAND MOBILE NETWORK	Q.1000–Q.1099
INTERWORKING WITH SATELLITE MOBILE SYSTEMS	Q.1100–Q.1199
INTELLIGENT NETWORK	Q.1200–Q.1699
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2000	Q.1700–Q.1799
SPECIFICATIONS OF SIGNALLING RELATED TO BEARER INDEPENDENT CALL CONTROL (BICC)	Q.1900–Q.1999
BROADBAND ISDN	Q.2000–Q.2999
General aspects	Q.2000–Q.2099
Signalling ATM adaptation layer (SAAL)	**Q.2100–Q.2199**
Signalling network protocols	Q.2200–Q.2299
Common aspects of B-ISDN application protocols for access signalling and network signalling and interworking	Q.2600–Q.2699
B-ISDN application protocols for the network signalling	Q.2700–Q.2899
B-ISDN application protocols for access signalling	Q.2900–Q.2999

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Q.2111

B-ISDN ATM adaptation layer – Service specific connection oriented protocol in a multilink and connectionless environment (SSCOPMCE)

Amendment 3

API for SSCOPMCE over Ethernet and UDP port number

Summary

This amendment specifies an Application Programming Interface in C++ for the protocol engine described in ITU-T Rec. Q.2111 Annex E (SSCOP in a Multi-link and Connectionless Environment when operating over Ethernet).

It also assigns the UDP port number for use with SSCOPMCE above UDP in arrangements as specified in Annexes C and D. This UDP port number may be used together with a port number out of the dynamic/private range (values from 49152 through 65535).

Source

Amendment 3 to ITU-T Recommendation Q.2111 (1999) was approved by ITU-T Study Group 11 (2001-2004) under the ITU-T Recommendation A.8 procedure on 14 October 2003.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

ITU-T Recommendation Q.2111

B-ISDN ATM adaptation layer – Service specific connection oriented protocol in a multilink and connectionless environment (SSCOPMCE)

Amendment 3

API for SSCOPMCE over Ethernet and UDP port number

1) Clause 2.2 – Bibliography

Add the following reference:

[24] ISO/IEC 14882:2003, *Programming languages – C++.*

2) Clause 5.3 – Modes of operation

Modify the last sentence at the end of the paragraph immediately following Figure 2 and add another sentence to read:

In addition, Annex F provides an Application Programming Interface (API) for SSCOPMCE over Ethernet, specified in Ada. Furthermore, Annex G provides an Application Programming Interface (API) for SSCOPMCE over Ethernet, specified in C++.

3) Clause C.3.2.1 – Description of the UDP upper interface

i) Modify "Source Port" at the end and add another sentence to read:

When present, the numeric value for SSCOPMCE above UDP is either "VALUE TO BE ASSIGNED BY IANA" or value out of the dynamic/private range (values from 49152 through 65535), according to the environment where SSCOPMCE is used.

ii) Modify "Destination Port" at the end and add another sentence to read:

The numeric value for SSCOPMCE above UDP is either "VALUE TO BE ASSIGNED BY IANA" or value out of the dynamic/private range (values from 49152 through 65535), according to the environment where SSCOPMCE is used.

4) Clause D.3.2.1 – Description of the UDP upper interface

i) Modify "Source Port" at the end and add another sentence to read:

When present, the numeric value for SSCOPMCE above UDP is either "VALUE TO BE ASSIGNED BY IANA" or value out of the dynamic/private range (values from 49152 through 65535), according to the environment where SSCOPMCE is used.

ii) Modify "Destination Port" at the end and add another sentence to read:

The numeric value for SSCOPMCE above UDP is either "VALUE TO BE ASSIGNED BY IANA" or value out of the dynamic/private range (values from 49152 through 65535), according to the environment where SSCOPMCE is used.

5) Annex G

Add new Annex G (API for SSCOPMCE over Ethernet) as follows:

Annex G

C++ API for SSCOPMCE over Ethernet

G.1 Introduction

Annex E specifies the deployment of SSCOPMCE on top of the connectionless service provided by IEEE 802.3 Ethernet networks. The primary driver for the configuration is to realize an open-systems databus for closed-loop systems.

Applications can utilize the following services of SSCOPMCE through the SAP offered by the SSCF at the UNI [12]:

- Unacknowledged transfer of data;
- Assured transfer of data;
- Transparency of transferred information;
- Establishment and release of connections for assured transfer of data.

Whereas the main body of the Recommendation and Annex E contain the specifications necessary to develop a product based on an Ethernet network interface card, this annex specifies an application programming interface (API) to the SAP. The reason for specifying an API is to drive development tool and/or real-time operating system vendors to offer a standard, open and familiar interface for software developers to take advantage of the network capabilities offered by an Ethernet-based databus.

G.2 Objectives of the Ethernet Databus API

The Ethernet Databus API is relatively small and self-contained, allowing a programmer to access SSCOPMCE services when such services operate over an Ethernet datalink layer. Two objectives were used in designing an API:

1) The API should be based on the notion of sockets, which has been widely used in the majority of existing network APIs for desktop and real-time operating systems. Sockets essentially treat each network connection as a stream into which bytes can be written-to or read-from, allowing them to be an extension of familiar file I/O concepts.
2) The API should include provisions for exception-handling in order to manage run-time errors.

G.3 Overview of the Ethernet Databus API Implementation

The Ethernet Databus API is written in the C++ programming language. The choice of C++ is based on its growing use in aerospace and defense systems, one of the application areas driving the specification of Annex E. Consequently, a C++-based API will permit the migration of existing system architectures toward an Ethernet-based databus. In addition, new system architectures may be based on it. Such an API will also offer a standard programming interface for use with an Ethernet-based databus.

The C++-based API defines the following types (objects):

- **EtherAddress**: Represents an Ethernet address.
- **EtherSocket**: Implements a client-side socket that utilizes the assured data transfer capabilities of SSCOPMCE. Data is transported in one or more sequenced-data (SD) PDUs within Ethernet frames.
- **EtherTag**: Contains the attributes associated with the 802.1 tag type [22].
- **EtherServerSocket**: Implements a server-side socket that utilizes the assured data transfer capabilities of SSCOPMCE. Data is transported in one or more sequenced-data (SD) PDUs within Ethernet frames.
- **Datagram**: Creates a datagram referring to an unnumbered user data (UD) PDU.
- **DatagramSocket**: Creates a socket to send or receive a datagram.
- **MulticastSocket**: Creates a multicast socket to send or receive a datagram. Data is transported in one or more unnumbered user data (UD) PDUs. Multicast operation is based on the GARP Multicast Registration Protocol (GMRP) [21].

The fact that only a few types are defined is largely due to the streamlined mapping of protocol layers allowed in Annex E. From a definition viewpoint, these types, and the associated operations on these types, are contained in the package Ethernet Databus. A driver associated with a network interface card must be compliant with it. From an implementation viewpoint, these types are designated as private, and, like the specification of associated operations, are outside the scope of this Recommendation. This has been done to allow flexibility in the implementation and evolution of the API.

G.4 Summary of C++ library definition

The following is a summary of the Ethernet Databus library:

Library <EthernetDatabus>

```
class EtherAddress {

public:
       char *addr;
       char *getOUI();
       char *getLocal();
       boolean isGroupAddress();
       char *getHostAddress();
       char **getAllHostAddresses();
       };


class EtherTag {

public:
      EtherTag(int cos);
      EtherTag(double vlan);
      EtherTag(int cos, boolean cfi, double vlan);
      int get_cos();
      boolean get_cfi();
      double get_vlan();
        };


class EtherSocket {
```

```
public:
        EtherSocket(const EtherAddress &host, int port);
        EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherAddress &interface, int localPort);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherTag &tag, const EtherAddress &interface,
                    int localPort);
        EtherAddress getEtherAddress();
        int getPort();
        int getLocalPort();
        EtherAddress getLocalAddress();
        istream &getInputStream();
        ostream &getOutputStream();
        void close();
        };


class EtherServerSocket{

public:
        EtherServerSocket(int port);
        EtherServerSocket(int port, int queueLength);
        EtherServerSocket(int port, int queueLength, const EtherAddress
                          &bindAddress);
        EtherSocket accept();
        void close();
        EtherAddress getEtherAddress();
        int getLocalPort();
        };

class Datagram {

public:

        // for receiving datagrams
        Datagram(unsigned char *buffer[ ], int length);
        Datagram(unsigned char *buffer[ ], int offset, int length);

        // for sending datagrams
        Datagram(unsigned char *data[ ], int length, const EtherAddress
                 &destination, int port);
        Datagram(unsigned char *data[ ], int offset, int length,
                 const EtherAddress &destination, int port);

        EtherAddress getAddress();
        int getPort();
        unsigned char **getData();
        int getLength();
        int getOffset();
        setData(unsigned char **data);
        setData(unsigned char **data, int offset, int length);
        void setAddress(const EtherAddress &remote);
        void setPort(int port);
        void setLength(int length);
        };

class DatagramSocket {

public:
        DatagramSocket();
        DatagramSocket(int port);
        DatagramSocket(int port, const EtherAddress &address);
```

```
        DatagramSocket(int port, const EtherAddress &address,
        const EtherTag &tag);

        void send(const Datagram &d);
        void receive(const Datagram &d);
        void close();
        void getLocalPort();
        void connect(const EtherAddress &host, int port);
        void disconnect();
        int getPort();
        EtherAddress getEtherAddress();
        };

class MulticastSocket:  public DatagramSocket {

public:
        MulticastSocket();
        MulticastSocket(int port);
        void joinGroup(const EtherAddress &address);
        void leaveGroup(const EtherAddress &address);
        void send(const Datagram &d);
        void setInterface(const EtherAddress &address);
        EtherAddress getInterface();
        };
```

G.5 Description of C++ library definition

The following is a detailed description of each of the classes and their associated member functions.

G.5.1 class EtherAddress

This class represents an Ethernet address.

```
class EtherAddress {

public:
       char *addr;
       char *getOUI();
       char *getLocal();
       boolean isGroupAddress();
       char *getHostAddress();
       char **getAllHostAddresses();
       };
```

Variables

addr

```
char *addr;
```

The six bytes of an Ethernet address, highest-order first. The address is formatted as a string describing the bytes in hex notation, e.g., "347B0046A8CE".

Methods

getOUI

```
char *getOUI();
```

Returns the first three bytes of an Ethernet address: the Organizationally Universal Identifer.

Returns:

The OUI part of the address, as a string describing the bytes in hex notation, e.g., "347B00".

getLocal

```
char *getLocal();
```

Returns the last three bytes of an Ethernet address: the locally assigned part.

Returns:
The locally assigned part of the address, as a string describing the bytes in hex notation, e.g., "46A8CE".

isGroupAddress

```
boolean isGroupAddress();
```

Determines whether the Ethernet address is a group address, if the first bit of the highest order byte is zero

Returns:
True if the address is a group address, false otherwise

getHostAddress

```
char *getHostAddress();
```

Returns the numeric Ethernet address associated with a host.

Returns:
A single Ethernet address, as a string describing the bytes in hex notation, e.g., "3487A8CE".

Throws: UnknownHostException
if no Ethernet address for the host could be found

getAllHostAddresses

```
char **getAllHostAddresses();
```

Returns an array of the addresses associated with a multi-homed host

Returns:
An array of Ethernet addresses, with a pointer to the first address

Throws: UnknownHostException
If no Ethernet address for the host could be found

G.5.2 class EtherTag

This class contains the attributes associated with the 802.1 tag type.

```
class EtherTag {

public:
      EtherTag(int cos);
      EtherTag(double vlan);
      EtherTag(int cos, boolean cfi, double vlan);
      int get_cos();
      boolean get_cfi();
      double get_vlan();
        };
```

Constructor

EtherTag

```
EtherTag(int cos);
```

Sets the CoS field of the 802.1 tag. The VLAN field is set to a default of all zeroes. The CFI field is set to a default value of zero.

Parameters:
cos – class of service

Throws: IllegalArgumentException
if one or more fields are improperly set.

EtherTag

```
EtherTag(double vlan);
```

Sets the CoS field of the 802.1 tag. The VLAN field is set to a default of all zeroes. The CFI field is set to a default value of zero.

Parameters:
vlan – vlan identifier

Throws: IllegalArgumentException
if one or more fields are improperly set.

EtherTag

```
EtherTag(double vlan);
```

Sets the VLAN field of the 802.1 tag. The CoS field is set to a default of all zeroes. The CFI field is set to a default value of zero.
Parameters:
vlan – vlan identifier
Throws: IllegalArgumentException
if one or more fields are improperly set.

EtherTag

```
EtherTag(int cos, boolean cfi, double vlan);
```

Sets all the fields of the 802.1 tag.
Parameters:
cos – class of service
cfi – canonical format identifier
vlan – virtual LAN identifier
Throws: IllegalArgumentException
if one or more fields are improperly set.

Methods

get_cos

```
int get_cos();
```

Returns the value of the CoS field in the 802.1 tag.
Returns:
the class of service

get_cfi

```
int get_cfi();
```

Returns the value of the CFI field in the 802.1 tag.
Returns:
the canonical format identifier

get_cfi

```
int get_vlan();
```

Returns the value of the VLAN field in the 802.1 tag.
Returns:
the vlan identifier

G.5.3 class EtherSocket

This class creates sockets that utilize the assured data transfer capabilities of SSCOPMCE. Data is transported in one or more sequenced-data (SD) PDUs within Ethernet frames.

```
class EtherSocket {

public:
        EtherSocket(const EtherAddress &host, int port);
        EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherAddress &interface, int localPort);
        EtherSocket(const EtherAddress &host, int port,
                    const EtherTag &tag, const EtherAddress &interface,
                    int localPort);
        EtherAddress getEtherAddress();
        int getPort();
        int getLocalPort();
        EtherAddress getLocalAddress();
        istream &getInputStream();
        ostream &getOutputStream();
```

```
  void close();
  };
```

Constructors

EtherSocket

```
EtherSocket(const EtherAddress &host, int port);
```

Creates a socket to the specified port on the specified host and tries to connect.
Parameters:
host – destination host address
port – destination port
Throws: IOException
if an I/O error occurs while creating the socket.

EtherSocket

```
EtherSocket(const EtherAddress &host, const EtherTag &tag, int port);
```

Creates a socket to the specified port on the specified host and tries to connect.
Parameters:
host – destination host address
tag – 802.1 tag
port – destination port
Throws: IOException
if an I/O error occurs while creating the socket.

EtherSocket

```
EtherSocket(const EtherAddress &host, int port,
    const EtherAddress &interface, int localPort);
```

Creates a socket to the specified port on the specified host and tries to connect. It connects to the host and port specified in the first two arguments, and from the local network interface and port specified in the last two arguments.
Parameters:
host – destination host address
port – destination port
interface – local address
localPort – local port
Throws: IOException
if an I/O error occurs while creating the socket.

EtherSocket

```
EtherSocket(const EtherAddress &host, int port,
        const EtherTag &tag, const EtherAddress &interface,
        int localPort);
```

Creates a socket to the specified port on the specified host and tries to connect. It connects to the host and port specified in the first two arguments, and from the local network interface and port specified in the last two arguments.
Parameters:
host – destination host address
port – destination port
tag – 802.1 tag
interface – local address
localPort – local port
Throws: IOException
if an I/O error occurs while creating the socket.

Methods

getEtherAddress

```
EtherAddress getEtherAddress();
```

Returns the remote host the socket is connected to or, if the connection is now closed, which host the socket was connected to when it was connected.
Returns:
the remote Ethernet address to which the socket is connected

getPort

```
int getPort();
```

Returns the port the socket is, or was or will be, connected to on the remote host.
Returns:
the port connected to on the remote host

getLocalPort

```
int getLocalPort();
```

Returns the port number for the local host.
Returns:
the local port number

getLocalAddress

```
EtherAddress getLocalAddress();
```

Gets the local address to which the socket is bound.
Returns:
the local address

getInputStream

```
istream &getInputStream();
```

Returns an input stream for this socket.
Returns:
a reference to an input stream for reading bytes from this socket.
Throws: IOException
if an I/O error occurs while creating the output stream.

getOutputStream

```
ostream &getOutputStream();
```

Returns an output stream for this socket.
Returns:
a reference to an output stream for writing bytes to this socket.
Throws: IOException
if an I/O error occurs while creating the output stream.

close

```
void close();
```

Closes the socket.
Throws: IOException
if an I/O error occurs while closing the socket.

G.5.4 class EtherServerSocket

This class implements server sockets.

```
class EtherServerSocket{

public:
        EtherServerSocket(int port);
        EtherServerSocket(int port, int queueLength);
        EtherServerSocket(int port, int queueLength, const EtherAddress
                          &bindAddress);
        EtherSocket accept();
        void close();
        EtherAddress getEtherAddress();
        int getLocalPort();
        };
```

Constructors

EtherServerSocket

```
EtherServerSocket(int port);
```

Creates a server socket on the port specified by the argument.

Throws: BindException
if the socket cannot be created and bound to the requested port, or if another server socket is already using the requested port

EtherServerSocket
```
EtherServerSocket(int port, int queueLength);
```
Creates a server socket on the specified port with the specified queue length for incoming connection requests.
Throws: BindException
if the socket cannot be created and bound to the requested port, or if another server socket is already using the requested port

EtherServerSocket
```
EtherServerSocket(int port, int queueLength, const EtherAddress
                  &bindAddress);
```
Creates a server socket on the specified port with the specified queue length to hold incoming connection requests; the socket binds only to the specified Ethernet address.
Throws: BindException
if the socket cannot be created and bound to the requested port, or if another server socket is already using the requested port

Methods

accept
```
EtherSocket accept();
```
Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
Throws: IOException
if an I/O error occurs while waiting for a connection.

close
```
void close();
```
Closes this socket.
Throws: IOException
if an I/O error occurs while closing the socket.

getEtherAddress
```
EtherAddress getEtherAddress();
```
Returns the local address of this server socket.
Returns:
the local address.

getLocalPort
```
int getLocalPort();
```
Determines the local port being listened on.
Returns:
the local port number

G.5.5 class Datagram

This class creates datagram referring to an unnumbered user data (UD) PDU.

```
class Datagram {

public:

        // for receiving datagrams
        Datagram(unsigned char *buffer[ ], int length);
        Datagram(unsigned char *buffer[ ], int offset, int length);

        // for sending datagrams
        Datagram(unsigned char *data[ ], int length, const EtherAddress
```

```
            &destination, int port);
    Datagram(unsigned char *data[ ], int offset, int length,
            const EtherAddress &destination, int port);

    EtherAddress getAddress();
    int getPort();
    unsigned char **getData();
    int getLength();
    int getOffset();
    setData(unsigned char **data);
    setData(unsigned char **data, int offset, int length);
    void setAddress(const EtherAddress &remote);
    void setPort(int port);
    void setLength(int length);
    };
```

Constructors

Datagram

`Datagram(unsigned char *buffer[ ], int length);`

Creates a datagram object for receiving data. The received datagram's data is stored in `buffer` until the appropriate UD PDU is filled or until `length` bytes have been written into the buffer.
Parameters:
buffer – array of bytes
length – number of bytes
Throws: IllegalArgumentException
if the specified length overflows the buffer

Datagram

`Datagram(unsigned char *buffer[ ], int offset, int length);`

Creates a datagram object for receiving data. The received datagram's data is stored in `buffer`, beginning at `buffer[offset]`, until the appropriate UD PDU is filled or until `length` bytes have been written into the buffer.
Parameters:
buffer – array of bytes
offset – offset, in bytes
length – number of bytes
Throws: IllegalArgumentException
if the specified length overflows the buffer

Datagram

```
Datagram(unsigned char *data[ ], int length, const EtherAddress
    &destination, int port);
```

Creates a datagram for sending data. The datagram is filled with `length` bytes of data. The `destination` points to the host the datagram is to be delivered to; the `port` is the destination port on that host.
Parameters:
data – array of bytes
length – number of bytes
destination – destination address
port – destination port
Throws: IllegalArgumentException
if the length is greater than the size of the data array

Datagram

```
Datagram(unsigned char *data[ ], int offset, int length, const
    EtherAddress &destination, int port);
```

Creates a datagram for sending data. The datagram is filled with `length` bytes of data starting at `offset`. The `destination` points to the host the datagram is to be delivered to; the `port` is the destination port on that host.
Parameters:
data – array of bytes
offset – offset, in bytes

length – number of bytes
destination – destination address
port – destination port
Throws: IllegalArgumentException
if the length is greater than the size of the data array

Methods

getAddress

```
EtherAddress getAddress();
```

Returns the address of the remote host from which the datagram was received.
Returns:
the remote host address

getPort

```
int getPort();
```

Returns the remote port from which the datagram was received.
Returns:
the remote port number

getData

```
unsigned char **getData();
```

Returns a byte array containing the data from the datagram.
Returns:
array of bytes

getLength

```
int getLength();
```

Returns the number of bytes in the datagram.
Returns:
number of bytes

getOffset

```
int getOffset();
```

Returns the point in the array returned by `getData()` where the data from the datagram begins.
Returns:
point in array where data begins

setData

```
setData(unsigned char **data);
```

Changes the payload of the datagram.

setData

```
setData(unsigned char **data, int offset, int length);
```

Sends data in `length` pieces beginning at `offset`.

setAddress

```
void setAddress(const EtherAddress &remote);
```

Changes the destination address of a datagram.

setPort

```
void setPort(int port);
```

Changes the port a datagram is addressed to.

setLength

```
void setLength(int length);
```

Changes the number of bytes in the internal buffer so datagrams are not truncated between receptions.

G.5.6 class DatagramSocket

This class creates a socket to send or receive a datagram.

```
class DatagramSocket {

public:
        DatagramSocket();
        DatagramSocket(int port);
        DatagramSocket(int port, const EtherAddress &address);
        DatagramSocket(int port, const EtherAddress &address,
        const EtherTag &tag);

        void send(const Datagram &d);
        void receive(const Datagram &d);
        void close();
        void getLocalPort();
        void connect(const EtherAddress &host, int port);
        void disconnect();
        int getPort();
        EtherAddress getEtherAddress();
        };
```

Constructors

DatagramSocket

`DatagramSocket();`

Creates a socket bound to an anonymous port. The same socket may be used to receive datagrams that a server sends back to it.
Throws: SocketException
if the socket cannot be created.

DatagramSocket

`DatagramSocket(int port);`

Creates a socket that listens for incoming datagrams on a specific port, specified by the `port` argument.
Parameters:
port – listening port
Throws: SocketException
if the socket cannot be created.

DatagramSocket

`DatagramSocket(int port, const EtherAddress &address);`

Creates a socket that listens for incoming datagrams on a specific port and network interface. This constructor is especially useful for a multi-homed host.
Parameters:
port – listening port
address – Ethernet address of the host
Throws: SocketException
if the socket cannot be created.

DatagramSocket

```
DatagramSocket(int port, const EtherAddress &address, const EtherTag
                &tagattr);
```

Creates a socket that listens for incoming datagrams on a specific port and network interface. This constructor is especially useful for a multi-homed host.
Parameters:
port – listening port
address – Ethernet address of the host
tagattr – 802.1 tag
Throws: SocketException
if the socket cannot be created.

Methods

send

```
void send(const Datagram &d);
```

Sends a single datagram dp over the network using this datagram socket.
Parameters:
d – datagram object
Throws: IOException
if datagram to be sent is larger than can be supported by the native software

receive

```
void receive(const Datagram &d);
```

Receives a single datagram from the network and stores it in the datagram dp.
Parameters:
d – datagram object
Throws: IOException
If there's a problem receiving the data

close

```
void close();
```

Frees the port occupied by the socket.

getLocalPort

```
void getLocalPort();
```

Returns the local port on which the socket is listening.
Returns:
the local port

connect

```
void connect(const EtherAddress &host, int port);
```

Enables the capability to send datagrams to and receive datagrams from the specified remote host on the specified remote port.

disconnect

```
void disconnect();
```

Disables the capability of the socket so that it can send datagrams to, and receive datagrams from, any host and port.

getPort

```
int getPort();
```

Returns the remote port to which the socket is connected.
Returns:
the remote port used by the connection; otherwise, a -1 is returned if the socket is not connected.

getEtherAddress

```
EtherAddress getEtherAddress();
```

Returns the address of the remote host to which the socket is connected.
Returns:
the address of the remote host; otherwise, a null is returned if the socket is not connected.

G.5.7 class MulticastSocket

This class creates a multicast socket. Data is transferred using the unacknowledged data transfer capabilities of SSCOPMCE.

```
class MulticastSocket:  public DatagramSocket {

public:
        MulticastSocket();
        MulticastSocket(int port);
        void joinGroup(const EtherAddress &address);
        void leaveGroup(const EtherAddress &address);
```

```
  void send(const Datagram &d);
  void setInterface(const EtherAddress &address);
  EtherAddress getInterface( );
  };
```

Constructors

MulticastSocket

```
MulticastSocket( );
```

Creates a multicast socket bound to an anonymous port. A recipient replies to the same port.
Throws: SocketException
if the socket cannot be created

MulticastSocket

```
MulticastSocket(int port);
```

Creates a multicast socket on a specific port.
Parameters:
port – source port
Throws: SocketException
if the socket cannot be created, e.g., if the port is already in use

joinGroup

```
void joinGroup(const EtherAddress &address);
```

Once a multicast socket is created, this method allows it to join a multicast group.
Parameters:
address – Ethernet address
Throws: IOException
if the address is not a group address

leaveGroup

```
void leaveGroup(const EtherAddress &address);
```

Once a multicast socket has joined a group, it can leave it by calling this method.
Parameters:
address – Ethernet address
Throws: IOException
if the address is not a group address

send

```
void send(const Datagram &d);
```

Sends a datagram over the multicast socket by calling this method inherited from the datagram class.
Parameters:
address – Ethernet address
Throws: IOException
If the datagram is larger than can be supported by the native software

setInterface

```
void setInterface(const EtherAddress &address);
```

Associates a particular network interface for multicast use on a multi-homed host.
Parameters:
address – Ethernet address
Throws: SocketException
if the address does exist on the local machine

getInterface

```
EtherAddress getInterface( );
```

Gets the address of the interface in use.
Returns:
the address in use

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure, Internet protocol aspects and Next Generation Networks
Series Z	Languages and general software aspects for telecommunication systems

Geneva, 2004