UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

# UIT-T Q.2111

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

**Amendement 2**
(04/2002)

SÉRIE Q: COMMUTATION ET SIGNALISATION

RNIS à large bande – Couche d'adaptation ATM de signalisation (SAAL)

---

# Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

## Amendement 2: interface API pour le protocole SSCOPMCE sur réseau Ethernet

Recommandation UIT-T Q.2111 – Amendement 2

---

RECOMMANDATIONS UIT-T DE LA SÉRIE Q

**COMMUTATION ET SIGNALISATION**

| | |
|---|---|
| SIGNALISATION DANS LE SERVICE MANUEL INTERNATIONAL | Q.1–Q.3 |
| EXPLOITATION INTERNATIONALE AUTOMATIQUE ET SEMI-AUTOMATIQUE | Q.4–Q.59 |
| FONCTIONS ET FLUX D'INFORMATION DES SERVICES DU RNIS | Q.60–Q.99 |
| CLAUSES APPLICABLES AUX SYSTÈMES NORMALISÉS DE L'UIT-T | Q.100–Q.119 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 4 | Q.120–Q.139 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 5 | Q.140–Q.199 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 6 | Q.250–Q.309 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R1 | Q.310–Q.399 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R2 | Q.400–Q.499 |
| COMMUTATEURS NUMÉRIQUES | Q.500–Q.599 |
| INTERFONCTIONNEMENT DES SYSTÈMES DE SIGNALISATION | Q.600–Q.699 |
| SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 7 | Q.700–Q.799 |
| INTERFACE Q3 | Q.800–Q.849 |
| SYSTÈME DE SIGNALISATION D'ABONNÉ NUMÉRIQUE N° 1 | Q.850–Q.999 |
| RÉSEAUX MOBILES TERRESTRES PUBLICS | Q.1000–Q.1099 |
| INTERFONCTIONNEMENT AVEC LES SYSTÈMES MOBILES À SATELLITES | Q.1100–Q.1199 |
| RÉSEAU INTELLIGENT | Q.1200–Q.1699 |
| PRESCRIPTIONS ET PROTOCOLES DE SIGNALISATION POUR LES IMT-2000 | Q.1700–Q.1799 |
| SPÉÇIFICATIONS DE LA SIGNALISATION RELATIVE À LA COMMANDE D'APPEL INDÉPENDANTE DU SUPPORT | Q.1900–Q.1999 |
| RNIS À LARGE BANDE | Q.2000–Q.2999 |
| Aspects généraux | Q.2000–Q.2099 |
| **Couche d'adaptation ATM de signalisation (SAAL)** | **Q.2100–Q.2199** |
| Protocoles du réseau sémaphore | Q.2200–Q.2299 |
| Aspects communs des protocoles d'application du RNIS-LB pour la signalisation d'accès, la signalisation de réseau et l'interfonctionnement | Q.2600–Q.2699 |
| Protocoles d'application du RNIS-LB pour la signalisation de réseau | Q.2700–Q.2899 |
| Protocoles d'application du RNIS-LB pour la signalisation d'accès | Q.2900–Q.2999 |

*Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.*

# Recommandation UIT-T Q.2111

## Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

## Amendement 2
## Interface API pour le protocole SSCOPMCE sur réseau Ethernet


**Résumé**

Le présent amendement à la Rec. UIT-T Q.2111 définit une interface de programmation d'application API (*application programming interface*) pour le protocole SSCOPMCE sur réseau Ethernet. Cette interface doit faciliter l'intégration du protocole SSCOPMCE dans le système de communication utilisant Ethernet.



**Source**

L'Amendement 2 de la Recommandation Q.2111 de l'UIT-T, élaboré par la Commission d'études 11 (2001-2004) de l'UIT-T, a été approuvé le 13 avril 2002 selon la procédure définie dans la Résolution 1 de l'AMNT.

## AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

## NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

## DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT avait été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

## TABLE DES MATIÈRES

# Recommandation UIT-T Q.2111

# Couche d'adaptation ATM du RNIS-LB – Protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE)

## Amendement 2
## Interface API pour le protocole SSCOPMCE sur réseau Ethernet

### 1) Paragraphe 2.2 – Bibliographie

*Ajouter la référence suivante:*

[23] ISO/IEC 8652:1995, *Technologies de l'information – Langages de programmation – Ada.*

### 2) Paragraphe 4 – Abréviations

*Ajouter l'abréviation suivante:*

API interface de programmation d'application (*application programming interface*)

### 3) Paragraphe 5.3 – Modes de fonctionnement

*Ajouter la phrase suivante à la fin de l'alinéa qui suit immédiatement la Figure 2:*

De plus, l'Annexe F définit une interface de programmation d'application (API, *application programming interface*) pour le protocole SSCOPMCE sur Ethernet.

### 4) Annexe F

*Ajouter la nouvelle Annexe F (Interface API pour le protocole SSCOPMCE sur Ethernet) comme suit:*

## Annexe F

## Interface API pour le protocole SSCOPMCE sur réseau Ethernet

### 1 Introduction

L'Annexe E de la présente Recommandation spécifie la mise en place du protocole en mode avec connexion propre au service dans un environnement avec liaisons multiples et sans connexion (SSCOPMCE, *service specific connection oriented protocol in a multilink and connectionless environment*) au sommet du service sans connexion assuré par les réseaux Ethernet IEEE 802.3. Il s'agit essentiellement pour la configuration de réaliser un bus de données pour systèmes ouverts destiné aux systèmes à boucle fermée.

Les applications peuvent utiliser les services SSCOPMCE suivants via les points SAP offerts par la fonction SSCF au niveau de l'interface UNI [12]:

- transfert de données sans accusé de réception;
- transfert de données garanti;

- transparence des informations transférées;
- établissement et libération des connexions pour le transfert de données garanti.

Le corps principal de la Recommandation et son Annexe E contiennent les spécifications nécessaires à l'élaboration d'un produit fondé sur une carte d'interface réseau Ethernet, en revanche la présente annexe spécifie une interface de programmation d'application (API) avec le point SAP. La spécification d'une interface API a pour objet d'amener les fournisseurs de systèmes d'exploitation à proposer une interface normalisée, ouverte et familière aux concepteurs de logiciels afin de tirer parti des capacités de réseau offertes par un bus de données Ethernet.

## 2 Objectifs de l'interface API avec bus de données Ethernet

Cette interface API avec bus de données Ethernet est relativement compacte et autonome, elle permet au programmeur d'accéder aux services SSCOPMCE lorsque ces services opèrent sur une couche Liaison de données Ethernet. La conception d'une interface API devait répondre aux deux objectifs suivants:

1) elle devait être fondée sur la notion de connecteur qui est très utilisée dans la plupart des interfaces API de réseau pour systèmes d'exploitation en temps réel des ordinateurs de bureau. Dans les connecteurs chaque connexion de réseau est essentiellement considérée comme un flux dans lequel on peut écrire ou lire des octets, ce qui permet de les considérer comme une extension des concepts familiers d'E/S de fichiers;
2) elle devait permettre le traitement des exceptions afin remédier aux erreurs de temps de traitement.

## 3 Mise en œuvre de l'interface API avec bus de données Ethernet: aperçu général

L'interface API avec bus de données Ethernet est programmée en langage Ada 95 [3]. Ce langage a été retenu en raison de son utilisation très répandue dans les systèmes aérospatiaux et les systèmes de défense, applications qui ont, entre autres, influencé l'élaboration de l'Annexe E/Q.2111. Par conséquent, un interface API en Ada permettra le passage des architectures de systèmes existantes à une architecture à bus de données Ethernet. De plus les nouvelles architectures de systèmes pourront être fondées sur ce bus de données. Ces interfaces API seront dotées d'une interface de programmation normalisée destinée à être utilisée avec un bus de données Ethernet.

L'interface API en Ada définit les types (objets) suivants:

- **EtherAddress**: représente une adresseEthernet.
- **EtherSocket**: implémente un connecteur côté client qui utilise les capacités de transfert garanti du protocole SSCOPMCE. Les données sont transportées dans une ou plusieurs unités PDU de données en séquence (SD, *sequenced-data*) dans les trames Ethernet.
- **EtherTag**: contient les attributs associés au type d'étiquette 802.1 [22].
- **EtherServerSocket**: implémente un connecteur côté serveur qui utilise les capacités de transfert garanti du protocole SSCOPMCE. Les données sont transportées dans une ou plusieurs unités PDU de données en séquence (SD) dans les trames Ethernet.
- **Datagram**: crée un datagramme renvoyant à une unité PDU de données d'utilisateur (UD, *user data*) numérotée.
- **DatagramSocket**: crée un connecteur pour envoyer et recevoir un datagramme.
- **MulticastSocket**: crée un connecteur de multidiffusion pour envoyer et recevoir un datagramme. Le données sont transportées dans une ou plusieurs PDU de données d'utilisateur (UD) numérotées. L'opération de multidiffusion est fondée sur le protocole d'enregistrement de multidiffusion (GMRP, *GARP multicast registration protocol*).

Le faible nombre de types qui sont définis tient au nouveau mappage autorisé dans l'Annexe E/Q.2111. Du point de vue des définitions, ces types, ainsi que les opérations associées sur ces types, sont contenus dans le progiciel Ethernet Databus. Un pilote associé à une carte d'interface de réseau doit lui être conforme. Du point de vue de la mise en œuvre, ces types sont désignés comme privés et, tout comme la spécification des opérations associées, sortent du cadre de la présente Recommandation. On dispose ainsi d'une certaine souplesse d'implémentation et d'évolution de l'interface API.

## 4 Récapitulatif des définitions contenues dans le progiciel Ada

Ce qui suit est un récapitulatif des définitions contenues dans le progiciel Ethernet Databus:

```
package Ethernet Databus is

     type EtherAddress is private;
     type EtherAddresses is (POSITIVE range <>) of  EtherAddress;

     type EtherTag is private;
     type COS_TYPE is mod 2**3;
     type VLAN_TYPE is mod 2**12;

     type EtherSocket is private;
     type PORT_TYPE is mod 2**16;

     type EtherServerSocket is private;

     type Datagram is private;
     type BYTE is mod 2**8;
     type BYTE_ARRAY is array (POSITVE range <>) of BYTE;

     type DatagramSocket is private;

     type MulticastSocket is private;

     -- EtherAddress

     function getAddress(addr: EtherAddress) return STRING;
     function getOUI(addr: EtherAddress) return STRING;
     function getLocal(addr: EtherAddress) return STRING;
     function isGroupAddress(addr: EtherAddress) return BOOLEAN;
     function getLocalAddress return EtherAddress;
     function getLocalAddresses return EtherAddresses;

     -- EtherTag

     procedure makeEtherTag(cos: in COS_TYPE) return EtherTag;
     procedure makeEtherTag(vlan: in VLAN_TYPE) return EtherTag;
     procedure makeEtherTag(cos: in COS_TYPE;
                            cfi: in BOOLEAN;
                            vlan: in VLAN_TYPE)
                            return EtherTag;
     function get_cos(tag: EtherTag) return COS_TYPE;
     function get_cfi(tag: EtherTag) return BOOLEAN;
     function get_vlan(tag: EtherTag) return VLAN_TYPE;

     -- EtherSocket

     function makeethersocket(host: etheraddress;
                            port: port_type)
                            return ethersocket;
     function makeEtherSocket(host: EtherAddress;
                            tag: EtherTag;
                            port: PORT_TYPE)
                            return  EtherSocket;
     function makeEtherSocket(host: EtherAddress;
                            port: PORT_TYPE;
                            interface: EtherAddress;
```

```
                          localPort: PORT_TYPE)
                          return EtherSocket;
function makeEtherSocket(host: EtherAddress;
                          port: PORT_TYPE;
                          tag: EtherTag;
                          interface: EtherAddress;
                          localPort: PORT_TYPE)
                                  return EtherSocket;
function getEtherAddress(socket: EtherSocket)
                          return EtherAddress;
function getPort(socket: EtherSocket)  return PORT_TYPE;
function getLocalPort(socket: EtherSocket)  return PORT_TYPE;
function getlocaladdress(socket: ethersocket)
                          return etheraddress;
function getInputStream(socket: EtherSocket)
                          return STREAM_ACCESS;
function getOutputStream(socket: EtherSocket)
                          return STREAM_ACCESS;
procedure close(socket: in EtherSocket);

-- EtherServerSocket

function makeEtherServerSocket(port: PORT_TYPE)
                                  return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                  tag: EtherTag)
                                  return EtherServerSocket;
function  makeEtherServerSocket(port: PORT_TYPE;
                                  queueLength: POSITIVE)
                                  return EtherServerSocket;
function  makeEtherServerSocket(port: PORT_TYPE;
                                  queueLength: POSITIVE;
                                  tag: EtherTag)
                                  return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                  queueLength: POSITIVE;
                                  bindAddress: EtherAddress)
                                  return EtherServerSocket;
function makeEtherServerSocket(port: PORT_TYPE;
                                  queueLength: POSITIVE;
                                  tag: EtherTag;
                                  bindAddress: EtherAddress)
                                  return EtherServerSocket;
function accept(socket: EtherServerSocket)
                                  return EtherSocket;
procedure close(socket: in EtherServerSocket);
function getEtherAddress(socket: EtherServerSocket)
                                  return EtherAddress;
function getLocalPort(socket: EtherServerSocket)
                                  return PORT_TYPE;
function getTag(socket: EtherServerSocket) return EtherTag;

-- DATAGRAM

-- pour les datagrammes reçus

function makeDatagram (buffer: BYTE_ARRAY;
                       length: POSITIVE)
                       return Datagram;

function makeDatagram (buffer: BYTE_ARRAY;
                       offset: NATURAL;
                       length: POSITIVE)
                       return Datagram;

-- pour l'envoi des datagrammes

function makeDatagram (data: BYTE_ARRAY;
                       offset: NATURAL;
                       length: POSITIVE)
                       return Datagram;
```

```
function makeDatagram (data: BYTE_ARRAY;
                       length: POSITIVE;
                       destination: EtherAddress;
                       port: PORT_TYPE)
                       return Datagram;

function getAddress(d: Datagram) return EtherAddress;
function getPort(d: Datagram) return PORT_TYPE;
function getData(d: Datagram) return BYTE_ARRAY;
function getLength(d: Datagram) return POSITIVE;
function getOffset(d: Datagram) return NATURAL;
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY);
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY;
                  offset: in NATURAL;
                  length: in POSITIVE);
procedure setAddress(d: in Datagram;
                     remote: in EtherAddress);
procedure setPort(d: in Datagram ;
                  port: in PORT_TYPE);
procedure setLength(d: in Datagram;
                    length: in POSITIVE);

-- DatagramSocket

function makeDatagramSocket return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                            tag: EtherTag)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                     address: EtherAddress)
                            return DatagramSocket;

function makeDatagramSocket(port: PORT_TYPE;
                             tag: EtherTag;
                             address: EtherAddress;
                             return DatagramSocket;

procedure send(socket: in DatagramSocket;
               d: in Datagram);
procedure receive(socket: in DatagramSocket;
                  d: in out Datagram);
procedure close(socket: in DatagramSocket);
function getLocalPort(socket: DatagramSocket) return PORT_TYPE;
procedure connect(socket: in DatagramSocket;
             host: in EtherAddress;
                  port: in PORT_TYPE);
procedure disconnect(socket: in DatagramSocket);
function getPort(socket: DatagramSocket) return PORT_TYPE;
function getEtherAddress(socket: DatagramSocket)
                         return EtherAddress;
function getTag(socket: DatagramSocket) return EtherTag;

-- MulticastSocket

function makeMulticastSocket return MulticastSocket;
function makeMulticastSocket(port: PORT_TYPE)
                             return MulticastSocket;
function makeMulticastSocket(port: PORT_TYPE;
                             tag: EtherTag)
                             return MulticastSocket;

procedure joinGroup(socket: in MulticastSocket;
                    address: in EtherAddress);
procedure leaveGroup(socket: in MulticastSocket;
                     address: in EtherAddress);
```

```
    procedure setInterface(socket: in MulticastSocket;
                           address: in EtherAddress);
    function getInterface(socket: MulticastSocket)
                           return EtherAddress;

    procedure send(socket: in MulticastSocket;
                   d: in Datagram);
    procedure receive(socket: in MulticastSocket;
                      d: in out Datagram);
    procedure close(socket: in MulticastSocket);
    function getLocalPort(socket: MulticastSocket)
                           return PORT_TYPE;
    procedure connect(socket: in MulticastSocket;
                      host: in EtherAddress;
                      port: in PORT_TYPE);
    procedure disconnect(socket: in MulticastSocket);
    function getPort(socket: MulticastSocket) return PORT_TYPE;
    function getEtherAddress(socket: MulticastSocket)
                             return EtherAddress;
    function getTag(socket: MulticastSocket) return EtherAddress;

    -- Exceptions

    UnknownHostException: exception;
    IlegalArgumentException: exception;
    BindException: exception;
    IOException: exception;
    SocketException: exception;

private

    -- dépend de l'implémentation

end Ethernet Databus;
```

## 5 Description des définitions continues dans le progiciel Ada

Ce qui suit est une description détaillée de chaque sous-programme.

### 5.1 Sous-programmes associés à EtherAddress

getAddress

```
function getAddress(addr: EtherAddress) return STRING;
```

Renvoie une adresse Ethernet complète de 48 bits

**Paramètres**:

addr – adresse Ethernet

**Renvoie:**

une seule adresse Ethernet, sous forme d'une chaîne décrivant les octets en notation hexadécimale, par exemple: "3407A4CE0000"

getOUI

```
function getOUI(addr: EtherAddress) return STRING;
```

Renvoie les trois premiers octets d'une adresse Ethernet, à savoir l'identificateur OUI (*organizationally universal identifier*).

**Paramètres**:

addr – adresse Ethernet

**Renvoie:**

la partie OUI de l'adresse, sous forme d'une chaîne décrivant les octets en notation hexadécimale. Exemple: "3407A4"

getLocal

```
function getLocal(addr: EtherAddress) return STRING;
```

Renvoie les trois derniers octets d'une adresse Ethernet, à savoir la partie assignée localement.

**Paramètres**:

addr – adresse Ethernet

**Renvoie:**

partie de l'adresse assignée localement sous forme d'une chaîne décrivant les octets en notation hexadécimale. Exemple: "CE0000"

isGroupAddress

```
function isGroupAddress(addr: EtherAddress) return BOOLEAN;
```

Détermine si l'adresse Ethernet est une adresse de groupe, c'est-à-dire si le premier bit de l'octet d'ordre le plus élevé est zéro

**Paramètres**:

addr – adresse Ethernet

**Renvoie:**

"True" si l'adresse est une adresse de groupe, "false" dans le cas contraire

getLocalAddress

```
function getLocalAddress(addr: EtherAddress) return STRING;
```

Renvoie l'adresse associée à l'hôte local.

**Paramètres**:

addr – adresse Ethernet locale

**Renvoie:**

une adresse Ethernet

**Envoie**: UnknownHostException

lorsque aucune adresse Ethernet pour l'hôte n'a pu être trouvée

getAllHostAddresses

```
function getLocalAddresses return EtherAddresses;
```

Renvoie un tableau des adresses associées à un hôte fortement connecté

**Renvoie:**

Un tableau d'adresses Ethernet

**Envoie**: UnknownHostException

Lorsque aucune adresse Ethernet pour l'hôte n'a pu être trouvée

## 5.2 Sous-programmes EtherTag

makeEtherTag

```
function makeEtherTag(cos: in COS_TYPE) return EtherTag;
```

Positionne le champ CoS de l'étiquette 802.1. Le champ VLAN prend la valeur par défaut qui est constituée de zéros uniquement. Le champ CFI prend la valeur par défaut qui est zéro.

**Paramètres**:

cos – classe de service

makeEtherTag

```
function makeEtherTag(vlan: in VLAN_TYPE) return EtherTag;
```

Positionne le champ VLAN de l'étiquette 802.1. Le champ CoS prend la valeur par défaut qui est constituée de zéros uniquement. Le champ CFI prend la valeur par défaut qui est zéro.

**Paramètres**:

vlan – identificateur de vlan

makeEtherTag

```
function makeEtherTag(cos: in COS_TYPE;
            cfi: in BOOLEAN;
                 vlan: in VLAN_TYPE)
                    return EtherTag;
```

positionne tous les champs de l'étiquette 802.1.

**Paramètres**:

cos – classe de service

cfi – identificateur de format canonique

vlan – identificateur de LAN virtuel

get_cos

```
function get_cos(tag: EtherTag) return COS_TYPE;
```

Renvoie la valeur du champ CoS dans l'étiquette 802.1.

**Paramètres**:

tag – étiquette 802.1

**Renvoie**:

la classe de service

get_cos

```
function get_cfi(tag: EtherTag) return BOOLEAN;
```

Renvoie la valeur du champ CFI dans l'étiquette 802.1.

**Paramètres**:
tag – étiquette 802.1
**Renvoie**:
l'identificateur de format canonique

get_vlan

```
function get_vlan(tag: EtherTag) return VLAN_TYPE;
```

Renvoie la valeur du champ VLAN dans l'étiquette 802.1.
**Paramètres**:
tag – étiquette 802.1
**Renvoie**:
l'identificateur vlan

## 5.3 Sous-programmes EtherSocket

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         port: PORT_TYPE)
                        return EtherSocket;
```

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion.
**Paramètres**:
host – adresse de l'hôte de destination
port – port de destination
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         tag: EtherTag;
                         port: PORT_TYPE)
                         return EtherSocket;
```

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente d'établir une connexion.
**Paramètres**:
host – adresse de l'hôte de destination
tag – étiquette 802.1
port – port de destination
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         port: PORT_TYPE;
                         interface: EtherAddress;
                         localPort: PORT_TYPE)
                         return EtherSocket;
```

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente de réaliser une connexion. Il se connecte à l'hôte et au port spécifiés dans les deux premiers arguments, et depuis l'interface de réseau et le port locaux spécifiés dans les deux derniers arguments.
**Paramètres**:
host – adresse de l'hôte de destination
port – port de destination
interface – adresse locale
localPort – port local
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur

makeEtherSocket

```
function makeEtherSocket(host: EtherAddress;
                         port: PORT_TYPE,
                         tag: EtherTag;
                         interface: EtherAddress;
                         localPort: PORT_TYPE)
                         renvoie EtherSocket;
```

Crée un connecteur vers le port spécifié sur l'hôte spécifié et tente de réaliser une connexion. Il se connecte à l'hôte et au port spécifiés dans les deux premiers arguments, et depuis l'interface de réseau et le port locaux spécifiés dans les deux derniers arguments.
**Paramètres**:
host – adresse de l'hôte de destination
port – port de destination
tag – étiquette 802.1
interface – adresse locale
localPort – port local
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du connecteur

getEtherAddress

```
function getEtherAddress(socket: EtherSocket)
                              return EtherAddress;
```

Renvoie l'adresse de l'hôte auquel le connecteur est relié ou, si la connexion est maintenant fermée, l'adresse de l'hôte auquel le connecteur était relié lorsqu'il était relié.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
l'adresse distante Ethernet à laquelle le connecteur est relié

getPort

```
function getPort(socket: EtherAddress) return PORT_TYPE;
```

Renvoie le port auquel le connecteur est, était ou sera relié sur l'hôte distant.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
le port auquel le connecteur est relié sur l'hôte distant

getLocalPort

```
function getLocalPort(socket: EtherAddress) return PORT_TYPE;
```

Renvoie le numéro de port pour l'hôte local.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
le numéro de port local

getLocalAddress

```
function getLocalAddress(socket: EtherAddress)
                              return EtherAddress;
```

Prend l'adresse locale auquel le connecteur est lié.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
l'adresse locale

getInputStream

```
function getInputStream(socket: EtherSocket)
                             return STREAM_ACCESS;
```

Renvoie un flux d'entrée pour ce connecteur.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
une référence à un flux d'entrée pour lire les octets à partir de ce connecteur.
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du flux de sortie

getOutputStream

```
function getOutputStream(socket: EtherSocket)
                              return STREAM_ACCESS;
```

Renvoie un flux de sortie pour ce connecteur.
**Paramètres**:
socket – connecteur Ethernet
**Renvoie**:
une référence à une flux de sortie pour écrire les octets à partir de ce connecteur.

**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la création du flux de sortie

close

```
procedure close(socket: in EtherSocket);
```

Ferme le connecteur.
**Paramètres**:
socket – connecteur Ethernet
**Envoie**: IOException
lorsqu'une erreur d'E/S s'est produite pendant la fermeture du connecteur

## 5.4 Sous-programmes EtherServerSocket

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE)
                                    return EtherServerSocket;
```

Crée un connecteur de serveur sur le port spécifié par l'argument.
**Paramètres**:
port – port local
**Envoie**: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                                     tag: EtherTag)
                                   return EtherServerSocket;
```

Crée un connecteur de serveur sur le port, et sur la base de l'étiquette, spécifié par les arguments.
**Paramètres**:
port – port local
tag – étiquette 802.1
**Envoie**: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                           queueLength: POSITIVE)
                         return EtherServerSocket;
```

Crée un connecteur de serveur sur le port spécifié avec la longueur de file d'attente spécifiée (en octets) pour les demandes de connexion entrantes.
**Paramètres**:
port – port local
queueLength – longueur de la file d'attente
**Envoie**: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

makeEtherServerSocket

```
function  makeEtherServerSocket(port: PORT_TYPE;
                           queueLength: POSITIVE;
                                   tag: EtherTag)
                                 return EtherServerSocket;
```

Crée un connecteur de serveur sur le port spécifié avec la longueur de file d'attente spécifiée (en octets) pour les demandes de connexion entrantes.
**Paramètres**:
port – port local
queueLength – longueur de la file d'attente
tag – étiquette 802.1
**Envoie**: BindException
lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
```

```
                                queueLength: POSITIVE;
                                bindAddress: EtherAddress)
                                        return EtherServerSocket;
```

Crée un connecteur de serveur sur le port spécifié avec le longueur de file d'attente spécifiée pour retenir les demandes de connexion; le connecteur assure uniquement le lien avec l'adresse Ethernet spécifiée.

**Paramètres**:

port – port local

queueLength – longueur de la file d'attente

bindAddress – adresse avec laquelle assurer le lien

**Envoie**: BindException

lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

makeEtherServerSocket

```
function makeEtherServerSocket(port: PORT_TYPE;
                               queueLength: POSITIVE;
                                       tag: EtherTag;
                               bindAddress: EtherAddress)
                                      return EtherServerSocket;
```

Crée un connecteur de serveur sur le port spécifié avec le longueur de file d'attente spécifiée pour retenir les demandes de connexion; le connecteur assure uniquement le lien avec l'adresse Ethernet spécifiée

**Paramètres**:

port – local port

queueLength – longueur de la file d'attente

bindAddress – adresse avec laquelle assurer le lien

tag – étiquette 802.1

**Envoie**: BindException

lorsque le connecteur ne peut être créé et lié au port demandé, ou lorsqu'un autre connecteur de serveur utilise déjà le port demandé

accept

```
function accept(socket: EtherServerSocket) return EtherSocket;
```

Se met en attente d'une connexion à réaliser vers ce connecteur et l'accepte. Cette méthode provoque un blocage jusqu'à ce que la connexion soit réalisée.

**Paramètres**:

socket – connecteur de serveur Ethernet

**Envoie**: IOException

Lorsqu'une erreur d'E/S s'est produite pendant l'attente d'une connexion

close

```
procedure close(socket: in EtherServerSocket);
```

Ferme ce connecteur.

**Paramètres**:

socket – connecteur de serveur Ethernet

**Envoie**: IOException

lorsqu'une erreur d'E/S s'est produite pendant la fermeture du connecteur

getEtherAddress

```
function getEtherAddress(socket: EtherServerSocket)
      return EtherAddress;
```

Renvoie l'adresse locale de ce connecteur de serveur.

**Paramètres**:

socket – connecteur de serveur Ethernet

**Renvoie**:

l'adresse locale

getLocalPort

```
function getLocalPort(socket: EtherServerSocket) return PORT_TYPE;
```

Détermine le port local sur lequel l'écoute a lieu.

**Paramètres**:

socket – connecteur de serveur Ethernet

**Renvoie**:

le numéro de port local.

getTag

```
function getTag(socket: EtherServerSocket) return EtherTag;
```

Renvoie l'étiquette de ce connecteur de serveur.
**Paramètres**:
socket – connecteur de serveur Ethernet
**Renvoie**:
l'étiquette ou, lorsqu'il n'y a pas d'étiquette associée à ce connecteur, une valeur nulle

## 5.5 Sous-programme Datagram

makeDatagram

```
function makeDatagram(buffer: BYTE_ARRAY;
          length: POSITIVE)
                    return Datagram;
```

Crée un objet datagramme pour la réception des données. Les données du datagramme reçues sont stockées dans `buffer` jusqu'à ce que l'unité PDU UD appropriée ait été remplie ou jusqu'à ce que les octets `length` aient été écrits dans le tampon.
**Paramètres**:
buffer – tableau d'octets
length – nombre d'octets
**Envoie**: IllegalArgumentException
lorsque la longueur spécifiée provoque un débordement du tampon

makeDatagram

```
function makeDatagram(buffer: BYTE_ARRAY;
                offset: NATURAL;
                   length: POSITIVE)
                            return Datagram;
```

Crée un objet datagramme pour la réception des données. Les données du datagramme reçues sont stockées dans `buffer`, en commençant à `buffer[offset]`, jusqu'à ce que l'unité PDU UD appropriée ait été remplie ou jusqu'à ce que les octets `length` aient été écrits dans le tampon.
**Paramètres**:
buffer – tableau d'octets
offset – décalage, en octets
length – nombre d'octets
**Envoie**: IllegalArgumentException
Lorsque la longueur spécifiée provoque un débordement du tampon

makeDatagram

```
function makeDatagram(data: BYTE_ARRAY;
          offset: NATURAL;
          length: POSITIVE)
                        return Datagram;
```

Crée un datagramme pour l'envoi des données. Le datagramme contient des octets `length` des données. `destination` pointe vers le serveur auquel doit être remis le datagramme, le `port` est le port de destination sur ce serveur.
**Paramètres**:
data – tableau d'octets
length – nombre d'octets
destination – adresse de destination
port – port de destination
**Envoie**: IllegalArgumentException
lorsque la longueur est supérieure à la taille du tableau de données

makeDatagram

```
function makeDatagram(data: BYTE_ARRAY;
          length: POSITIVE;
           destination: EtherAddress;
                      port: PORT_TYPE)
                        return Datagram;
```

Crée un datagramme pour l'envoi des données. Le datagramme contient des octets `de longueur` des données commençant à `offset`. `destination` pointe vers le serveur auquel doit être remis le datagramme, le port est le port de destination sur ce serveur.
**Paramètres**:
data – tableau d'octets
offset – décalage en octets

length – nombre d'octets
destination – adresse de destination
port – port de destination
**Envoie**: IllegalArgumentException
lorsque la longueur est supérieure à la taille du tableau de données

getAddress

```
function getAddress(d: Datagram) return EtherAddress;
```

Renvoie l'adresse du serveur distant d'où provient le datagramme reçu.
**Paramètres**:
d – datagramme
**Renvoie**:
l'adresse du serveur distant

getPort

```
function getPort(d: Datagram) return PORT_TYPE;
```

Renvoie le port distant d'où provient le datagramme reçu.
**Paramètres**:
d – Datagramme
**Renvoie**:
le numéro du port distant

getData

```
function getData(d: Datagram) return BYTE_ARRAY;
```

Renvoie un tableau d'octets contenant les données provenant du datagramme.
**Paramètres**:
d – datagramme
**Renvoie**:
le tableau d'octets

getLength

```
function getLength(d: Datagram) return POSITIVE;
```

Renvoie le nombre d'octets du datagramme.
**Paramètres**:
d – datagramme
**Renvoie**:
le nombre d'octets

getOffset

```
function getOffset(d: Datagram) return NATURAL;
```

Renvoie le point du tableau renvoyé par `getData` où les données extraites du datagramme commencent.
**Paramètres**:
d – datagramme
**Renvoie**:
point dans le tableau où les données commencent

setData

```
procedure setData(d: in Datagram;
                  data: in BYTE_ARRAY);
```

modifie la charge utile du datagramme.
**Paramètres**:
d – datagramme
data – tableau d'octets

setData

```
procedure setData(d: in Datagram
                  data: in BYTE_ARRAY;
                  offset: in NATURAL;
                  length: in POSITIVE);
```

Envoie des données par tranches de longueur commençant à `offset`.
**Paramètres**:
d – datagramme
data – tableau d'octets
offset – décalage
length: taille de la tranche de données

setAddress

```
procedure setAddress(d: in Datagram;
                     remote: in EtherAddress);
```

Modifie l'adresse de destination d'un datagramme.
**Paramètres**:
d – datagramme
remote – adresse Ethernet distante

setPort

```
procedure setPort(d: in Datagram;
                  port: in PORT_TYPE);
```

Modifie le port vers lequel le datagramme envoyé.
**Paramètres**:
d – datagramme
port – port de destination

setLength

```
procedure setLength(d: in Datagram;
                    length: in POSITIVE);
```

Modifie le nombre d'octets dans le tampon interne de manière à ce que les datagrammes ne soient pas tronqués entre les réceptions.
**Paramètres**:
d – datagramme
length – Longueur en octets

## 5.6 Sous-programmes DatagramSocket

makeDatagramSocket

```
function makeDatagramSocket return DatagramSocket;
```

Crée un connecteur lié à un port anonyme. Le même connecteur peut être utilisé pour recevoir des datagrammes renvoyés par un serveur.
**Envoie**: SocketException
lorsque le connecteur ne peut être créé.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE)
                            return DatagramSocket;
```

Crée un connecteur qui écoute des datagrammes entrants sur un port spécifique, spécifié par l'argument port.
**Paramètres**:
port – port d'écoute
**Envoie**: SocketException
lorsque le connecteur ne peut être créé

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                            tag: EtherTag)
                           return DatagramSocket;
```

Crée un connecteur qui écoute des datagrammes entrants sur un port spécifique, spécifié par l'argument port, et l'étiquette spécifiée par l'argument tag.
**Paramètres**:
port – port d'écoute
tag – étiquette 802.1
**Envoie**: SocketException
lorsque le connecteur ne peut être créé.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                         address: EtherAddress)
                            return DatagramSocket;
```

Crée un connecteur qui écoute des datagrammes entrants sur un port et une interface de réseau spécifiques. Cette structure est spécialement utile pour un hôte fortement connecté.
**Paramètres**:
port – port d'écoute
address – adresse Ethernet du serveur

**Envoie**: SocketException
lorsque le connecteur ne peut être créé.

makeDatagramSocket

```
function makeDatagramSocket(port: PORT_TYPE;
                tag: EtherTag;
                        address: EtherAddress)
                           return DatagramSocket;
```

Crée un connecteur qui attend les datagrammes sur un port, une étiquette et une interface de réseau spécifiques. Cette structure est particulièrement utile pour un hôte fortement connecté.
**Paramètres**:
port – port d'écoute
tag – étiquette 802.1
address – adresse Ethernet de l'hôte
**Envoie**: SocketException
lorsque le connecteur ne peut être créé.

send

```
procedure send(socket: DatagramSocket;
                        d: in Datagram);
```

Envoie un seul datagramme dp sur le réseau en utilisant le connecteur de datagramme.
**Paramètres**:
socket – connecteur pour datagramme
d – objet datagramme
**Envoie**: IOException
lorsque la taille du datagramme à envoyer est supérieure à celle que le logiciel spécifique peut prendre en charge

receive

```
procedure receive(socket: in DatagramSocket;
                           d: in out Datagram);
```

Reçoit un seul datagramme du réseau et le stocke dans l'objet datagramme d.
**Paramètres**:
socket – connecteur pour datagramme
d – objet datagramme
**Envoie**: IOException
lorsqu'il y a un problème de réception des données

close

```
procedure close(socket: in DatagramSocket);
```

Libère le port occupé par le connecteur.
**Paramètres**:
socket – connecteur pour datagramme

getLocalPort

```
function getLocalPort(socket: DatagramSocket)
                            return PORT_TYPE;
```

Renvoie le port local sur lequel le connecteur est en attente (écoute).
**Paramètres**:
socket – connecteur pour datagramme
**Renvoie**:
le port local

connect

```
procedure connect(socket: DatagramSocket;
                        host: in EtherAddress;
                        port: in PORT_TYPE);
```

Active la capacité d'envoi et de réception de datagrammes vers ou en provenance de l'hôte distant spécifié sur le port distant spécifié.
**Paramètres**:
socket – connecteur pour datagramme
host – adresse Ethernet
port – port distant

disconnect

```
procedure disconnect(socket: in DatagramSocket);
```

Désactive la capacité du connecteur de sorte qu'il peut envoyer et recevoir des datagrammes vers ou en provenance d'un hôte ou d'un port quelconque.
**Paramètres**:
socket – connecteur de datagramme

getPort
```
function getPort(socket: DatagramSocket) return PORT_TYPE;
```
Renvoie le port distant auquel le connecteur est relié.
**Paramètres**:
socket – connecteur de datagramme
**Renvoie**:
le port distant utilisé par la connexion ou, si le connecteur n'est pas raccordé, une information nulle

getEtherAddress
```
function getEtherAddress(socket: DatagramSocket)
                              return EtherAddress;
```
Renvoie l'adresse de l'hôte distant auquel le connecteur est relié.
**Paramètres**:
socket – connecteur de datagramme
**Renvoie**:
l'adresse de l'hôte distant ou, si le connecteur n'est pas raccordé, une information nulle

getTag
```
function getTag(socket: DatagramSocket) return EtherTag;
```
Renvoie l'étiquette associée au connecteur.
**Paramètres**:
socket – connecteur de datagramme
**Renvoie**:
l'étiquette; ou, si aucune information n'est associée à ce connecteur, une information nulle

## 5.7 Sous-programmes MulticastSocket

makeMulticastSocket
```
function makeMulticastSocket return MulticastSocket;
```
Crée un connecteur de multidiffusion lié à un port anonyme. Le destinataire répond au même port.
**Envoie**: SocketException
en cas d'impossibilité de création du connecteur

makeMulticastSocket
```
function makeMulticastSocket(port: PORT_TYPE)
                                 return MulticastSocket;
```
Crée un connecteur de multidiffusion sur un port spécifique.
**Paramètres**:
port – port source
**Envoie**: SocketException
lorsque le connecteur ne peut être créé, par exemple, lorsque le port est déjà utilisé

makeMulticastSocket
```
function makeMulticastSocket(port: PORT_TYPE;
                               tag: EtherTag)
                              return MulticastSocket;
```
Crée un connecteur de multidiffusion sur un support spécifique en utilisant une étiquette spécifiée.
**Paramètres**:
port – port source
tag – étiquette 802.1
**Envoie**: SocketException
lorsque le connecteur ne peut être créé, par exemple, lorsque le port est déjà utilisé

joinGroup
```
procedure joinGroup(socket: MuticastSocket;
                    address: in EtherAddress);
```
Après la création d'un connecteur de multidiffusion, cette méthode lui permet de se joindre à un groupe de multidiffusion.
**Paramètres**:
socket – connecteur de multidiffusion
address – adresse Ethernet

**Envoie**: IOException
lorsque l'adresse n'est pas une adresse de groupe

leaveGroup

```
procedure leaveGroup(socket: MutlicastSocket;
                     address: in EtherAddress);
```

Après qu'un connecteur de multidiffusion se soit joint à un groupe, il peut le quitter en appelant cette méthode.
**Paramètres**:
socket – connecteur de multidiffusion
address – adresse Ethernet
**Envoie**: IOException
lorsque l'adresse n'est pas une adresse de groupe

setInterface

```
procedure setInterface(socket: MultiastSocket;
                       address: in EtherAddress);
```

Associe une interface de réseau particulière destinée à être utilisée pour la multidiffusion sur un hôte fortement connecté.
**Paramètres**:
Socket – connecteur de multidiffusion
address – adresse Ethernet
**Envoie**: SocketException
lorsque l'adresse n'existe pas dans la machine locale

getInterface

```
function getInterface(socket: MulticastSocket)
                       return EtherAddress;
```

Obtient l'adresse de l'interface en cours d'utilisation.
**Renvoie**:
l'adresse en cours d'utilisation

send

```
procedure send(socket: in MulticastSocket;
                     d: in Datagram);
```

Envoie un seul datagramme dp sur le réseau en utilisant ce connecteur de datagramme.
**Paramètres**:
socket – connecteur de multidiffusion
d – objet datagramme
**Envoie**: IOException
lorsque la taille du datagramme à envoyer est supérieure à celle que le logiciel spécifique peut prendre en charge

receive

```
procedure receive(socket: in MulticastSocket;
                        d: in out Datagram);
```

Reçoit un seul datagramme en provenance du réseau et le stocke dans le datagramme d.
**Paramètres**:
d – objet datagramme
**Envoie**: IOException
lorsqu'il y a un problème pour recevoir les données

close

```
procedure close(socket: in MulticastSocket);
```

Libère le port occupé par le connecteur.
**Paramètres**:
socket – connecteur de multidiffusion

getLocalPort

```
function getLocalPort(socket: MulticastSocket)
                       return PORT_TYPE;
```

Renvoie le port local sur lequel le connecteur est en position d'écoute.
**Paramètres**:
socket – connecteur de multidiffusion
**Renvoie**:
le port local

connect

```
procedure connect(socket: in MulticastSocket;
                  hôte: in EtherAddress;
                  port: in PORT_TYPE);
```

Active la capacité du connecteur de sorte qu'il peut envoyer et recevoir des datagrammes vers ou en provenance d'hôtes distants sur le port distant spécifié

**Paramètres**:

socket – connecteur de multidiffusion

host – adresse de l'hôte distant

port – port distant

disconnect

```
procedure disconnect(socket: in MulticastSocket);
```

Désactive la capacité du connecteur de sorte qu'il peut envoyer et recevoir des datagrammes vers ou en provenance d'un hôte ou d'un port quelconque.

**Paramètres**:

socket – connecteur de multidiffusion

getPort

```
function getPort(socket: MulticastSocket) return PORT_TYPE;
```

Renvoie le port distant auquel le connecteur est relié.

**Paramètres**:

socket – connecteur de multidiffusion

**Renvoie**:

le port distant utilisé pour cette connexion ou un "1" si le connecteur n'est pas relié

getEtherAddress

```
function getEtherAddress(socket: MulticastSocket)
                         return EtherAddress;
```

Renvoie l'adresse du serveur distant auquel le connecteur est relié.

**Paramètres**:

socket – connecteur de multidiffusion

**Renvoie**:

l'adresse du serveur distant ou, si le connecteur distant n'est pas raccordé, une information nulle

getTag

```
function getTag(socket: MulticastSocket) return EtherTag;
```

Renvoie l'étiquette associée au connecteur.

**Paramètres**

socket – connecteur de multidiffusion

**Renvoie**:

l'étiquette ou, si l'étiquette n'est pas associée à ce connecteur, une information nulle

## SÉRIES DES RECOMMANDATIONS UIT-T

| | |
|---|---|
| Série A | Organisation du travail de l'UIT-T |
| Série B | Moyens d'expression: définitions, symboles, classification |
| Série C | Statistiques générales des télécommunications |
| Série D | Principes généraux de tarification |
| Série E | Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains |
| Série F | Services de télécommunication non téléphoniques |
| Série G | Systèmes et supports de transmission, systèmes et réseaux numériques |
| Série H | Systèmes audiovisuels et multimédias |
| Série I | Réseau numérique à intégration de services |
| Série J | Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias |
| Série K | Protection contre les perturbations |
| Série L | Construction, installation et protection des câbles et autres éléments des installations extérieures |
| Série M | RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie et circuits loués internationaux |
| Série N | Maintenance: circuits internationaux de transmission radiophonique et télévisuelle |
| Série O | Spécifications des appareils de mesure |
| Série P | Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux |
| **Série Q** | **Commutation et signalisation** |
| Série R | Transmission télégraphique |
| Série S | Equipements terminaux de télégraphie |
| Série T | Terminaux des services télématiques |
| Série U | Commutation télégraphique |
| Série V | Communications de données sur le réseau téléphonique |
| Série X | Réseaux de données et communication entre systèmes ouverts |
| Série Y | Infrastructure mondiale de l'information et protocole Internet |
| Série Z | Langages et aspects généraux logiciels des systèmes de télécommunication |

Genève, 2002