EUROPEAN TELECOMMUNICATION STANDARD

DRAFT

pr **ETS 300 325**

January 1996

Second Edition

Source: ETSI TC-TE

Reference: RE/TE-02040

ICS: 33.080

Key words: API, CAPI, ISDN, PCI

Integrated Services Digital Network (ISDN); Harmonized Programmable Communication Interface (PCI); for ISDN

ETSI

European Telecommunications Standards Institute

ETSI Secretariat

Postal address: F-06921 Sophia Antipolis CEDEX - FRANCE
Office address: 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE
X.400: c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 92 94 42 00 - Fax: +33 93 65 47 16

Contents

Blank page

Foreword

This draft European Telecommunication Standard (ETS) has been produced by the Terminal Equipment (TE) Technical Committee of the European Telecommunications Standards Institute (ETSI), and is now submitted for the Public Enquiry phase of the ETSI standards approval procedure.

Transposition dates	
Date of adoption of this ETS:	DD Month YYYY
Date of latest announcement of this ETS (doa):	DD Month YYYY
Date of latest publication of new National Standard or endorsement of this ETS (dop/e):	DD Month YYYY
Date of withdrawal of any conflicting National Standard (dow):	DD Month YYYY

Introduction

The number of different Integrated Services Digital Network (ISDN) Programming Interfaces used by Terminal Equipment (TE) has hindered the development of applications using ISDN which, in turn, has proved a constraint to the usage of ISDN on modern Terminal Equipment.

This ETS defines the ETSI ISDN Application Programmable Interface (API), called ISDN Programmable Communication Interface (PCI). The ISDN PCI is a application interface for accessing and administering ISDN services.

It has been defined in order to provide a standard that TE providers should implement instead of providing their own programming interface. Thus allowing the portability of applications that use the ISDN PCI across a range of TE based on different operating systems.

The ISDN PCI has been defined with the Application Developer in mind and, where possible, eliminates the need for a detailed knowledge of ISDN. It has also been defined in such a manner that extensions provided to take advantage of future ISDN developments do not effect the operation of existing applications.

Although this ETS applies to the Euro-ISDN, other ISDNs and ISDN-like networks can easily be supported by this interface description.

In order to enhance the first edition of this ETS (March 1994), two profiles have been introduced.

Future extensions of services should be offered for both profiles. Those extensions should ensure backward compatibility with this ETS.

Blank page

1 Scope

This ETS specifies two profiles:

- Profile A covers the March 1994 version of ETS 300 325 including additional protocols and operating system extensions.

- Profile B describes an alternative interface access that is bit compatible to COMMON-ISDN-API (CAPI 2.0 - May 1995).

The PCI described in this ETS accesses and administers the following services:

- bearer services as defined in ETS 300 102-1 [2];
- supplementary services as defined in ETS 300 196 [7];
- Virtual Circuit (VC) or Permanent Virtual Circuit (PVC) Bearer Services on the B- and D-channels.

It:

- covers both basic and primary rate ISDN access;

- is independent of operating systems, hardware and programming languages. It provides language and operating system binding for common operating system environments;

- supports concurrent applications;

- supports concurrent protocol stacks related to data exchange;

- supports application access to multiple channels on multiple ISDN accesses;

- provides the Open Systems Interconnection (OSI) connection-mode network service as defined by ITU-T Recommendation X.213 [6] using the method defined in ISO 9574 [9];

- provides an interface for applications requiring direct control of ISDN services;

- shows the impact of security issues on the interface;

- has been defined to allow future extension of functionality;

- supports Dual Tone Multi Frequency (DTMF) access.

Further standards specify the method of testing and detailed application specific requirements to determine conformance based on this standard.

2 Normative references

This ETS incorporates by dated and undated references, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to these ETS only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred to applies.

[1] ETS 300 080 (1992): "Integrated Services Digital Network (ISDN); ISDN lower layer protocols for telematic terminals".

[2] ETS 300 102-1 (1990): "Integrated Services Digital Network (ISDN); User-network interface layer 3, Specifications for basic call control".

[3] ISO 8208 (1990): "Information technology; Data communications; X.25 Packet Layer Protocol for Data Terminal Equipment".

[4] ISO 7776 (1986): "Information Processing systems; Data communications; High-level data link control procedures; Description of the X.25 LAPB-compatible DTE data link procedures".

[5] ITU-T Recommendation Z.100 (1988): "Functional specification and description language (SDL)".

[6] ITU-T Recommendation X.213 (1992):"Information technology - Network service definition for Opens Systems".

[7] ETS 300 196 (1991): "Integrated Services Digital Network (ISDN); Generic Functional protocol for the support of supplementary services Digital Subscriber Signalling System No. one (DSS1) protocol".

[8] ITU-T Recommendation Q.931 (1993): "Digital subscriber Signalling System No. 1 (DSS1) - ISDN user-network interface layer 3 specification for basic call control".

[9] ISO 9574 (1989): "Information Technology - Telecommunications and information exchange between systems - Provision of the OSI connection-mode network service by packet mode terminal equipment connected to an integrated services digital network (ISDN)".

[10] Request For Comment (RFC) 1661: "The Point-to-Point Protocol (PPP)".

[11] Request For Comment (RFC) 1618: "PPP over ISDN".

[12] IBM publication: "IBM Synchronous Data Link Control Concepts" (GA27-3093).

[13] ITU-T Recommendation Q.921 (1993): "ISDN user-network interface - Data link layer specification".

[14] ITU-T Recommendation T.30 (1993): "Procedures for document facsimile transmission in the general switched telephone network".

[15] ITU-T Recommendation T.70 (1993): "Network-independent basic transport service for the telematic services".

[16] CCITT Recommendation T.90 (1992): "Characteristics and protocols for terminals for telematic services in ISDN".

[17] CCITT Recommendation V.110 (1992): "Support of data terminal equipments with V-Series type interfaces by an integrated services digital network".

[18] ITU-T Recommendation X.30 (1993): "Support of X.21, X.21 bis and X.20 bis based data terminal equipments (DTEs) by an integrated services digital network (ISDN)".

[19] ETS 300 097 (1992): "Integrated Services Digital Network (ISDN); Connected Line Identification Presentation (COLP) supplementary service; Digital Subscriber Signalling System No. one (DSS1) protocol part 1; Protocol implementation description".

[20] ITU-T Recommendation X.25: "Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packetmode and connected to public data networks by dedicated circuit".

For further references to publications, which are of interest when reading this ETS, refer to the bibliography contained in annex A.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this ETS, the following definitions apply:

address set: A set of parameters containing remote and local user layer or signalling addresses.

Administration Plane: A logical grouping of functionality for management of PUF-NAF dialogue as well as for access to local or network related NAF resources.

attribute set: Set of parameters driving user protocols and ISDN signalling.

B-channel: Logical ISDN channel for the use of data transfer.

Control Plane: Logical grouping of functionality for access of ISDN signalling.

controller: Hardware unit which gives access to an ISDN.

D-channel: Logical ISDN channel used for signalling and in some cases, for data transfer.

Euro-ISDN: ISDN offering services and interoperability in Europe as agreed upon in the "Memorandum of Understanding on the Implementation of European ISDN Service by 1992".

exchange function: PUF functionality realising the Exchange Mechanism.

exchange mechanism: Means provided for the PUF to interchange Messages with the NAF.

implementation: A version of a profile coded in hardware and/or software that provides the programmable interface to an application for a given Operating System.

ISDN access: Set of ISDN channels provided by a single Network Access Facility (NAF) to access ISDN services.

ISDN Programming Communication Interface (ISDN PCI): Network (ISDN) oriented software interface providing access provisions for programming network signalling and user data exchange.

message: Unit of information transferred through the interface between the Network Access Facility (NAF) and the PCI User Facility (PUF).

Network Access Facility (NAF): Functional unit located between the Profile A interface and the network related layers.

Network Connection Object (NCO): Abstract object within the NAF that shall be created by the PUF to gain access to network signalling or data.

Network Layer Message Access (NMA): Logical message access to ISDN network layer user protocols.

NULL layer: Describes an empty layer of the OSI reference model. Such a layer does not contain any functionality and passes requests and responses transparently to adjourning layers.

PCI User Facility (PUF): Functional unit using the Profile A interface to access a NAF. In fact, the local application using the interface.

signalling message access: Logical message access to signalling part of ISDN.

transparent message access: Logical message access to ISDN physical layer.

Type-Length-Value Coding (TLV Coding): Coding scheme used for binary presentation of Messages.

user connection: Connection accessible through User Plane functionality.

User Plane: Logical grouping of functionality for access of user protocols and data.

user protocol: Protocol running and conforming to User Plane functionality.

3.2 Abbreviations

For the purposes of this ETS, the following abbreviations apply:

API	Application Programming Interface
ASP	Abstract Service Primitive (i.e. OSI service primitive exchanged at a Service Access Point (SAP))
CAPI	COMMON-ISDN-API
CIP	Connection Identification Profile
CONS	COnnection-mode Network Service
ETS	European Telecommunication Standard
HDLC	High-Level Data Link Control
HLC	High Layer Compatibility
ISDN	Integrated Services Digital Network
LAP B	Link Access Procedure Balanced
LAP D	Link Access Procedure for D-channel
LLC	Low Layer Compatibility
MOU	Memorandum Of Understanding
N-SAP	Network layer - Service Access Point
NAF	Network Access Facility
NCCI	Network Control Connection Identifier
NCO	Network Connection Object
NMA	Network layer Message Access
PCI	Programming Communication Interface
PciMPB	Pci Message Parameter Block
PCO	Point of Control and Observation
PDU	Protocol Data Unit
Ph-SAP	Physical layer - Service Access Point
PLCI	Physical Link Connection Identifier
PUF	Programming Communication Interface User Facility
SAP	Service Access Point
SFF	Structured Fax File
T.70NL	T.70 Network Layer
T.90NL	T.90 Network Layer
TLV coding	Type-Length-Value coding (used for presentation of Profile A messages)
X.25 PLP	X.25 Packet Layer Protocol

4 General

4.1 Overview

With countries of the European Community committed by their Memorandum of Understanding to implement one standard for ISDN throughout Europe, it is a logical step forward to define an API that provides access to this ISDN. The goals of this API can be summarised as follows:

- provide access to the ISDN, while not prohibiting access to existing ISDN implementations;
- allow for standalone and distributed operation;
- provide an interface that is capable of supporting multiple applications;
- provide an interface capable of providing support for multiple ISDN accesses;
- provide an interface that supports both basic and primary rate access;
- provide an interface that is, as far as possible, operating system independent;
- define the interface in sufficient detail to ensure binary compatibility between different implementations within the same operating system on the same platform;
- allow access to supplementary services provided via the ISDN;
- provide support for physical devices such as telephones;
- good performance.

Both profiles can be seen as satisfying these goals and providing an interface that is highly suitable to:

- ISDN adapter manufacturers;
- ISDN application writers;
- ISDN users as a procurement requirement for selecting ISDN products and applications.

Both profiles in this ETS cover the same field of application with different information coding. Because of these differences, an application based on one of the profiles is not intended to interwork with the implementation of the other profile.

The most distinguishing characteristics of the two profiles are listed below:

Profile A:

a) supports a co-ordination function that abstracts the ISDN signalling by offering the ISO connection-mode network service (CONS) as defined in ITU-T Recommendation X.213 [6];

b) presents the information elements by converting network presentation to Type-Length-Value (TLV) coding more suitable for the application;

c) provides transparent access to the network signalling information elements allowing usage with any signalling protocol implementation.

Profile B:

1) is independent from the signalling protocol (e.g. also usable on non Q.931 based networks);

2) provides an interface which abstracts from protocol details not important for applications;

3) is bit-compatible with COMMON-ISDN-API Version 2.0 of May 1995.

4.2 Requirements

To ensure that applications can work with either profile, an implementation which provides the ISDN PCI interface shall implement both profiles.

Mandatory protocols are:

- Profile A: ISO 8208 [3], ETS 300 080 [1], Transparent byte oriented B-channel access;
- Profile B: High-Level Data Link control (HDLC), X.75.

If an implementation provides a non-mandatory protocol, it shall provide that protocol for both profiles.

An implementation for a specific Operating System shall be always provided for both profiles.

4.3 Reader's guide

Table 1 gives a descriptive list showing the full contents of this ETS.

Table 1: List of ETS contents

Clause Annex	Contains
Clause 1	Scope of this ETS. This describes what this ETS covers.
Clause 2	Normative references.
Clause 3	Definitions of the terms and Abbreviations used throughout this ETS.
Clause 4	General information about this ETS.
Clause 5	Description of Profile A of this ETS.
Clause 6	Description of Profile B of this ETS.
Clause 7	The exchange mechanism description for both profiles.
Annex A	Bibliography. Informative references useful for the understanding of this ETS for both profiles.
Annex B	The mapping between the Profile A messages and the underlying protocols.
Annex C	Details for NAFs providing external equipment support (telephony) (Profile A).
Annex D	Rules for use of the X.25 protocol (Profile A).
Annex E	Guidelines for NAF developers and manufacturers giving guidance for implementation and extension of the Profile A.
Annex F	Profile A implementation description: a template for description of the implementation for further conformance statements of PUF or NAF developers.
Annex G	Definition of standard profile. This annex provides the content of static attributes (Profile A).
Annex H	Sample coding in C language illustrating operating system specific implementation of the exchange mechanism (Profile A).
Annex J	C language illustrating TLV encoding/decoding example (Profile A).
Annex K	Sample Flow Chart diagrams of the Profile B.
Annex L	Description of the Structured Facsimile File Format used by the Profile B.
Annex M	Protocols list supported by Profile B.
Annex N	Development guidelines for Profile B.
Annex P	Profile B Implementation description.
Annex Q	Index of related topics for the Profile B.

5 Profile A

5.1 Reader's guidance and overview

5.1.1 Reader's guide

This ETS is intended for:

- software developers and implementors of applications by providing them with the definition of a simple, standardized and portable interface giving access to ISDN;

- manufacturers and developers of ISDN adapters and system software with the aim of providing a standardized programmable interface to ISDN communications;

- users of ISDN based software and management personnel by providing them with background information and selection criteria for choosing ISDN products and applications.

5.1.2 How to use this profile

Readers who:

- need a quick overview of the Profile A features and capabilities should read the overview provided in subclause 4.3. More detailed information about the architecture and functional description is provided in subclause 5.2. General User Protocol management can be found in subclause 5.6.1. Clause 3 provides useful information on definitions of terms and abbreviations used;

- intend to implement an application using this profile should first read subclauses 5.3, 5.4 and 5.5. Clauses 3 and 4 provide useful information on the definitions of terms, abbreviations and the implementation description used. For more detailed information on user protocols, subclause 5.6 and annex E should also be inspected. Detailed information on exchange method implementation are provided in clause 7 and in annex J.

- intend to build an ISDN adapter card or equipment should also first inspect subclauses 5.3, 5.4 and 5.5. Clause 3 and 4 provides useful information on the definitions of terms, abbreviations and implementation description used. For more detailed information on user protocols, subclause 5.6 should also be inspected. Annex E describes informative default configuration values and NAF diagrams. Detailed information on exchange method implementation are provided in clause 7 and in annex J.

The basic services subclauses provide:

- the definition of the functional model (subclause 5.2.1);
- encoding principles (subclause 5.2.2);
- the description of administrative and control messages and parameters (subclauses 5.4, 5.5 and 5.6);
- the exchange method definition (subclause 5.3);
- security aspects of Profile A (subclause 5.10).

The user protocol usage Management Architecture clause provides:

- the definition of message accesses (subclause 5.6.1.2);
- the list of supported protocols and the selection method (subclause 5.6.1.3);
- co-ordination function information (subclause 5.6.1.4);
- common Network Connection Object (NCO) selection criteria (subclause 5.6.1.5);
- error checking principles (subclause 5.6.1.6);
- AttributeSet contents (subclause 5.6.1.7).

For each supported protocol, this ETS provides, in order:

- a description of available user messages;
- a description of useful user parameters;
- the protocol state diagram;
- co-ordination function information;

- specific NCO selection criteria if it exists;
- specific error handling and codes;
- AttributeSet definition.

Annex E gives default protocol configuration values, if any, and shows NAF Specification and Description Language (SDL) diagrams describing most of the cases.

For each supported operating system, this ETS provides:

- application level implementation of exchange functions;
- NAF level implementation of exchange functions;
- NAF availability.

Annex J gives coding examples.

5.1.3 Functional overview

The basic model of PROFILE A consists of two entities: a service user called the PCI User Facility (PUF) and a service provider called the Network Access Facility (NAF). The PUF and the NAF interact by means of messages. Using these messages, the PUF requests the NAF to perform actions and to return the results to the PUF.

The interface to the NAF, depending on the underlying protocols supported, is divided into 3 planes:

- a Control Plane which provides access to the services offered by ISDN;

- a User Plane which provides access to the protocols used to transfer the data over connections established through the ISDN;

- an Administration Plane which provides the mechanisms that support the objects and identifiers required by the other two planes.

Each plane groups a distinctive set of functionality which is exchanged through the Profile A. The method for exchanging the information is called an "exchange mechanism" and is defined separately. This definition is generic in nature and may be applied to several operating systems. The adaptation of the exchange mechanism to a choice of specific, popular operating systems is covered in separate clauses.

Apart from the use of non-connection related facilities, e.g. security features and support of external equipment, the use of the Profile A is based upon the establishment of ISDN and user protocol connections. Within the PCI the concept of a Network Connection Object (NCO) is used to control these connections. NCOs are defined by the use of Administration Plane messages and used in both Control and User Plane messages to establish, use and remove connections.

5.1.4 Connection management

Performing ISDN connections leads firstly to set up the protocol related information required for this connection and then to unambiguously identify all messages attached to this connection. The specification provides a mechanism to link the attributes - static or dynamic - to the connection based on NCO management.

5.1.5 The planes

The Control Plane supports messages that allow the PUF to establish, control and remove connections, and to access the services provided by the ISDN. Seven classes of message are defined. The first class is associated with the basic call set-up and is mandatory for the NAF to provide while the other classes associated with overlapping dialling method, telephony supplementary services, user-to-user signalling and adjournment of calls, external equipment and DTMF are optional.

The Administration Plane is responsible for managing attribute sets, addresses and NCOs. It offers messages that provide information concerning the state of any external equipment that the NAF controls, such as a telephone. It also provides messages to manage the security features used on a particular connection. Four classes of message are defined. The first, associated with the basic operation of the

NAF, is defined as mandatory for the NAF to provide while the other classes, associated with security, manufacturer specific features and protocol modification for NCO are optional.

The User Plane provides messages that allow the use of underlying protocols. At present several sets of messages are defined. Three sets of messages are defined in the User Plane. One set allows access to User Plane protocols providing the OSI Network-layer service interface. The second one provides access to link-layer service interface. The last set provides a transparent interface where the PUF implements the protocol to be run over the connection.

5.1.6 Properties

Profile A defines that each NAF provide the PUF with a list of its static properties. These properties define the capabilities of the NAF generally and the resources to which it has access to in a particular configuration. It is through these sets of properties that the PUF is informed of the variety of messages the NAF supports within each plane. As an example, these properties may refer to the types of external equipment available or the type and number of ISDN channel available.

5.1.7 External equipment (i.e. telephony)

Profile A provides support of external equipment controlled by the NAF. The types of external equipment supported relate to various types of telephony equipment such as headsets and others. This support is achieved by treating the external equipment as a special type of transparent access so that when an ISDN connection is established (using the transparent protocol) the relevant ISDN channel for that external equipment is attached - this method is preferred rather than providing the PUF with User Plane messages. Administration Plane messages are provided to monitor the state of external equipment.

5.1.8 ISDN accesses and the multi-applications environment

The definition of Profile A supports various NAF - PUF configurations. It puts no constraints on the NAF implementation and it allows:

- a NAF to provide either only one or more than one ISDN accesses;
- a NAF to allow access to only one or more than one PUF concurrently;
- any number of NAFs to coexist within the same TE.

5.1.9 Exchange mechanism

The realisation of an interface across several distinct platforms (operating system) is often difficult to reconcile. Each operating system has a particular way of doing this and very often the same interface cannot be ported from one system to another. One way of going about solving this problem is by means of defining an exchange mechanism. The exchange mechanism abstracts the functionality between the interacting elements by means of functions. The Profile A interface defines seven functions that allow the registration, de-registration, and conversation of exchanges. During the registration phase, a function is usable to provide a list of the available NAFs within the system.

Messages passed to the NAF are copied before control is returned to the PUF. Once the PUF regains control, it is free to re-use any memory associated with the message. The memory allocation for data being transferred or received is performed by pointers inside the messages, this mechanism avoids making unnecessary copies. It is the responsibility of the PUF to provide memory for the NAF to place any messages. In order to assist the PUF in the provision of memory space, a signalling mechanism defined within each operating system is described; this mechanism consists of notifying the PUF when a message is available.

5.2 Message overview

5.2.1 Functional model

5.2.1.1 Introduction

Subclause 5.2.1 describes the functional model for Profile A. It introduces the architecture of Profile A and also describes the functionality of Profile A and the interactions between the entities located around the Profile A interface. Furthermore, it describes the sequencing of messages, to indicate in which way the entities may exchange information.

There is also a description of the identifiers involved in Profile A and the error mechanism it provides.

5.2.1.2 Architecture

Profile A is the specification of the communication interface inside TE which wishes to access an ISDN. Using this interface enables a higher layer entity to access the services of an ISDN network in a standardized way.

Profile A is a software interface between a service user and a service provider. As a software interface, Profile A consists of the specification of the interface and a description of the functionality which lies directly below the interface.

Profile A is an interface specification which is implemented in a actual computer environment. This environment imposes problems, e.g. associating the functional units and exchanging information between the entities. As a result, Profile A contains some functionality to deal with the problems of implementing it within a computer environment.

Two entities can be distinguished around Profile A. These are the service user and the service provider. These entities, along with Profile A and their information interchange are described in subclause 5.2.1.2.1.

5.2.1.2.1 Profile A and its components

The functional components relevant for the definition of Profile A and their relationship are shown in figure 1.

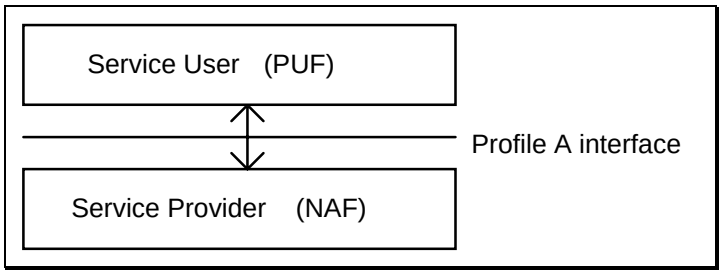


Figure 1: Functional picture of the Profile A interface with surrounding components

The functional components relevant for Profile A are:

PUF

Throughout this ETS, the term PCI User Facility (PUF) is used to refer to the user of the service. PUF refers to all the functional layers that use the interface to access the services of the ISDN.

NAF

The term Network Access Facility (NAF) is used to refer to the service provider. NAF refers to all elements which are necessary to provide access to the services of ISDN. These elements can be both software and hardware, no distinction is made. The NAF behaves as representing the services of one ISDN access.

Profile A

Profile A defines the interface located at the top of the NAF(s). Profile A defines a number of functions. First, Profile A allows for the association between the PUF and the NAF. After the PUF and NAF are associated, all the operations are performed by an information exchange mechanism. The exchange mechanism is another part of the functionality of Profile A. Figure 1 describes how the Profile A interface relates to the surrounding components. The arrow indicates the information flow.

Messages

Accessing the functionality described by Profile A is achieved by means of messages. The PUF and NAF use the functionality of the information exchange mechanism to exchange messages. The messages inform the entities of the operations to perform, or the results of performed operations.

5.2.1.2.2 Profile A architecture

Profile A has its own structure (described in figure 2). This structure consists of three planes, separating the functionalities. Each plane has its own set of messages. Profile A distinguishes the following planes:

- **Control Plane;**
  the Control Plane is related to the signalling part of a connection, which is via the NAF associated with the signalling in the ISDN D-channel. It covers the functionality provided by the service in the D-channel, such as connection control, control of service characteristics and supplementary services. Furthermore, the Control Plane is responsible for managing special equipment accessible through the Profile A interface.

- **User Plane;**
  the User Plane is related to the user connection, which may either be associated with a connection on the B-channel or a data connection on the D-channel. It is associated via the NAF with the functionality provided by the data services in the D- and B-channels, which consists of services for end-to-end data exchange.

- **Administration Plane;**
  the Administration Plane does not relate to the ISDN. It covers the required functionality for control and configuration of the Control and User Planes.

Figure 2 gives a representation of the three planes.

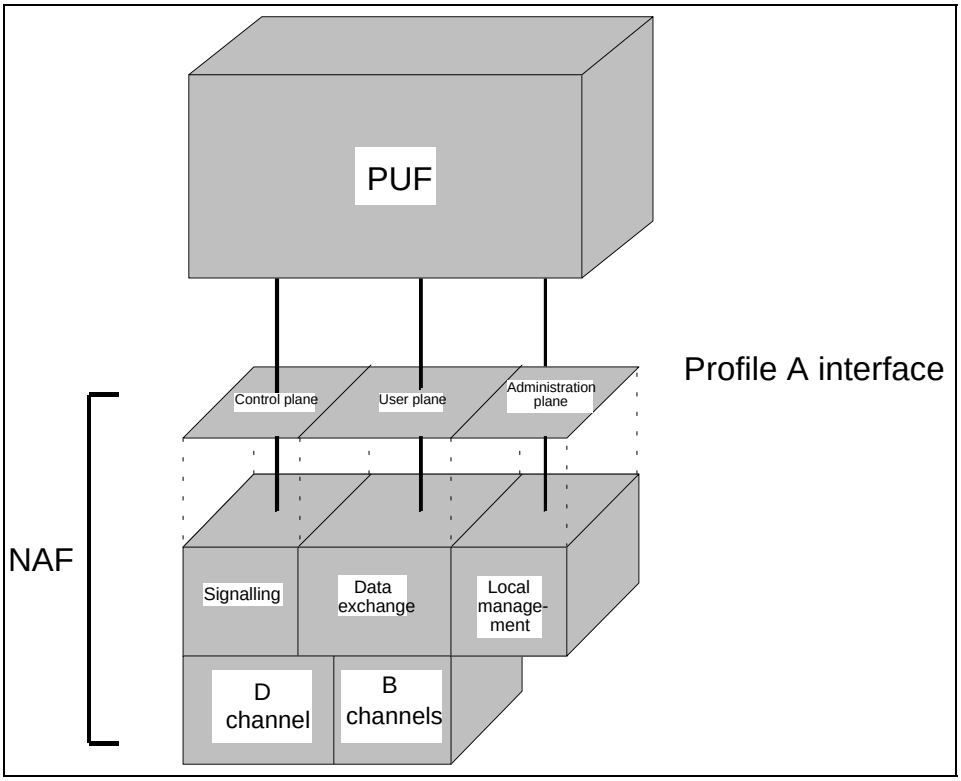


Figure 2: Relation between planes and ISDN

5.2.1.2.3 Co-ordination cases

Profile A provides for two mechanisms to co-ordinate the functionalities associated with the ISDN signalling and the user connection.

In the PUF co-ordination case, the PUF shall handle the establishment of a user connection by using the basic call control provided by the Control Plane. As a result of controlling the signalling connection, the PUF can use the supplementary services.

In the NAF co-ordination case, an abstraction is provided by a co-ordination function, which maps the primitives of CONS X.213 [6] in the User Plane according to the primitives of the Control Plane and User Plane protocols. A detailed description of the condition and procedures for the use of the co-ordination function is provided in subclause 5.6. Since the NAF manages the co-ordination between signalling and user connection, the PUF shall not access the Control Plane.

5.2.1.3 Functionality

5.2.1.3.1 Introduction

As described in subclause 5.2.1.2, Profile A functionality is provided by the three planes, with associated message sets to access the functionality. How the exchange of messages between PUF and NAF takes place is described in subclause 5.2.1.5.

In order to access ISDN signalling or data, the PUF shall request the NAF for the creation of a NCO. The creation and destruction of network connection is the main part of the functionality of the resource management.

After having performed this successfully, the PUF is in an "idle" state and may subsequently access ISDN signalling (except in case of NAF co-ordination) or transfer data. Subclauses 5.2.1.3.3.1 and 5.2.1.3.3.1 respectively, describe the functionality for connection management and data management.

5.2.1.3.2 Resource management

The resource management functionality is needed to be able to use Profile A for communication. Resource management contains functionality for local management. The functionality covers the management of:

- Network Connection Objects (NCOs);
- external equipment.

The Administration Plane of Profile A provides the functionality defined by the resource management.

The resource management evolves around the NCO, which is the object needed for subsequent communication. An NCO refers to an abstract object containing all relevant configuration information for one user connection. The configuration information for an NCO shall be assigned by the PUF using one of two principal methods:

a) referencing a standardized attribute set;
b) specifying all configuration information during NCO creation.

Method a) provides a simple way for the PUF to select appropriate configuration information by referencing a standardized attribute set identifier. However, this method is available at the cost of flexibility, since attribute sets are standardized and may only be used in the provided manner. Annex H gives the list of standardized attribute sets for the Control Plane.

Method b) gives the PUF the opportunity to specify configuration information for any special needs on its own. However, this involves a lot of details concerning D-channel and B-channel parameters and shall, therefore, be left for the sophisticated PUF-implementor.

5.2.1.3.2.1 Attribute sets

Attribute sets are used to keep together important parameters for configuring user protocols, for executing the ISDN signalling protocol and for collecting some management information relevant to the NCOs (statistics, cost, ...). User protocols and ISDN signalling are accessed through the functionality of the User Plane and the Control Plane. A collection of attribute sets exists for both planes. These sets are:

- signalling attribute set (related to the Control Plane);
- user protocol attribute set (related to the User Plane);
- administration attribute set (related to the Administration Plane).

The administration attribute set is not involved in the NCO creation but is only updated during the life of the NCO and can be accessed at any time through the resource management.

The resource management offers functionality to reference specific attribute sets when creating an NCO.

5.2.1.3.2.2 Network Connection Objects

The resource management functionality covers:

- the creation of a NCO;
- the grouping of NCOs;
- the information retrieval on a NCO.

A NCO is an abstract object created by the NAF in response to requests by the PUF prior to the establishment of a connection.

As a rule there is one NCO per connection, independent of which type of connection the NCO is related to. This can be a signalling connection or a connection for data transfer.

After the successful creation of a NCO, a unique identifier, the NCO identifier (NCOID), becomes available. This NCOID shall be supplied in subsequent operations regarding connection establishment and data transfer.

At the creation time of a NCO, the PUF can indicate that the newly created NCO should be grouped to another NCO already in existence.

The purpose of the grouping is to provide the ability to share a channel when using a network layer protocol, which allows the sharing of several logical connections on one physical channel. The sharing is reserved to one PUF.

The grouping functionality is User Plane protocol dependent. A detailed description of the condition and procedures for the use of the grouping functionality is provided in subclause 5.6.

The grouping of the NCOs is done by using the Group-ID. A unique Group-ID shall be returned on the successful creation of a NCO. This Group-ID can subsequently be supplied at the creation of an additional NCO, which shall then be grouped to the first NCO. If no Group-ID is supplied, the NCO shall not be grouped. The Group-ID is only guaranteed to be unique for the interaction between the PUF and the NAF.

As the GroupID is only unique for the PUF-NAF relation, multiple PUFs which access the same NAF cannot share the same connection.

For an incoming call, the NAF selects the appropriate NCOs and is then helped by the PUF to choose the unique one. This is done using the SelectorID, supplied at the creation of the NCO. This gives the PUF the opportunity to handle a list of NCOs that the NAF will exclusively deal with.

In case of a non co-ordinated NCO (C/U), User and Control Plane may have different directions. For example, the User Plane may be listening, while the Control Plane is calling.

5.2.1.3.2.3 Support of external equipment

Access to external equipment, such as telephones, is provided to the PUF through the functionality of the three planes.

As long as a NCO that specifies an external equipment in its configuration information exists, the NAF shall generate the appropriate Control Plane messages if the state of that external equipment changes.

The connection management (subclause 5.2.1.3.3) and data management (subclause 5.2.1.3.3) provide functionality to manage connections with these NCO.

Five types of external equipments are defined:

1) external equipment without telephony hook control. This type of external equipment only contains the transceivers. In this case, the PUF is in charge of managing the ISDN connection;

2) external equipment with telephony hook control. In this case, all telephony hook events are available at the interface level and the PUF is in charge of managing the ISDN connection;

3) external equipment with telephony hook control and which is able to manage the ISDN connection. In this case, all telephony hook control events are available at the Profile A;

4) external equipment with keypad and with or without telephony hook control. In this case all dialling events, all telephony hook control events are available at the Profile A and the PUF is in charge of managing the ISDN connection;

5) external equipment with keypad and with or without telephony hook control which is able to manage the ISDN connection. In this case, all dialling events, all telephony hook control events and information about the status of the communication are available at the Profile A.

All these types of external equipment are connected to the NAF by the means of a proprietary connection which is outside the scope of this ETS and provides to the PUF the availability or not of the external equipment.

In the case of type 4 and 5 external equipments, there can be two types of dialling:

- blocksending: one Control Plane message containing the complete destination address is provided to the PUF;

- overlap sending: one Control Plane message per key pressed is provided to the PUF. During a communication, DTMF codes can be sent via the keypad.

Type 3 external equipment is able to deal with incoming calls alone when the computer is off.

Type 5 external equipment is able to deal with incoming and outgoing calls when the computer is off.

Each action to the handset generates a Control Plane message to the PUF. Depending on the type of external equipment, different level of messages are sent to the PUF:

- for type 1 external equipment:
  - availability/unavailability;

- for type 2 and 3 external equipment:
  - availability/unavailability;
  - on-hook;
  - off-hook.

- for type 4 and 5 external equipment:
  - availability/unavailability;
  - on hook;
  - off hook;
  - a code representing the key pressed on the keypad in the case of an overlap sending;
  - a table of codes representing the complete destination address in the case of a block sending.

In the case of a type 2 and 3 external equipments and if the PUF has created a NCO that specifies an external equipment in its signalling attribute set, a connection which involves this external equipment may be established and breakdowns in different ways:

- case of the outgoing calls:
  - the user goes off-hook via the handset and the PUF issues the overlap or block dialling;
  - the PUF issues the overlap or block dialling and the user goes off-hook via the handset;

- case of incoming calls:
  - the user goes off-hook via the handset and the PUF receives a Control Plane message to inform it;
  - the PUF answers the incoming call and the user goes off-hook via the handset;

- case of local closedown:
  - the user on-hooks the handset and the PUF receives a Control Plane message to inform it;
  - the PUF releases the call and the user goes on-hook via the handset;

- case of remote closedown:
  - the PUF receives a Control Plane message and the user goes on-hook via the handset.

In the case of a type 4 and 5 external equipments and if the PUF created a NCO that specifies an external equipment in its signalling attribute set, a connection which involves this external equipment may be established and closed down in different ways:

- case of the outgoing calls:
  - the user goes off-hook via the handset and the PUF issues the overlap or block dialling;
  - the user goes off-hook via the handset which generates a Control Plane message to the PUF and uses the keypad of the external equipment to issue the overlap dialling. Each key pressed generates a Control Plane message to the PUF;
  - the user goes off-hook via the handset which generates a Control Plane message to the PUF and uses the keypad of the external equipment to issue the block dialling. The end of the destination address is detected by the means of a special key. A Control Plane message is generated to the PUF;
  - the PUF issues the overlap or block dialling and the user off-hooks the handset;

- case of the incoming calls:
  - the user goes off-hook via the handset and the PUF receives a Control Plane message to inform it of the off-hook state;
  - the PUF answers to the incoming call and the user goes off-hook via the handset;

- case of local breakdown:
  - the user goes on-hook via the handset and the PUF receives a Control Plane message to inform it of the on-hook state;
  - the PUF releases the call and the user goes on-hook via the handset;

- case of remote breakdown:
  - the PUF receives a Control Plane message and the user goes on-hook via the handset.

5.2.1.3.2.4 Support of security features

Access to security features below the Profile A interface is provided to the PUF through the functionality of the Administration Plane. The security features provided cover the use of security algorithms on connections.

The PUF can activate and deactivate security features on a specific connection by supplying the NCO of the connection in Administration Plane messages.

5.2.1.3.2.5 Support of manufacturer specific features

Access to manufacturer specific features is provided to the PUF through the functionality of the Administration Plane.

The PUF can access manufacturer specific features by using this functionality. It is a way to handle extra functionality not provided by Profile A.

The information exchanged between PUF and NAF is dependent of the implementation of the manufacturer specific feature and is, therefore, not covered in this ETS.

5.2.1.3.3 Connection management

The connection management functionality covers two aspects:

- the set-up and breakdown of physical connections;
- the access and usage of supplementary services.

The connection set-up and breakdown covers the basic functionality of the physical connection management. The supplementary services provide additional functionality related to the physical connection management.

The Control Plane described in Profile A provides the functionality defined by the physical connection management.

5.2.1.3.3.1 Connection set-up and removal

The only way for a PUF to achieve a connection is to enter the "idle" state by the creation of a NCO. Subsequently, it can perform a connection request or wait for a connection indication. After the connection is removed, the PUF returns to the "idle" state and can subsequently reuse the NCO for a new connection. The NCO becomes invalid if it is destroyed or if the PUF deregisters from the NAF.

At the creation of a NCO the PUF shall decide which type of connection is to be achieved. Profile A provides for access to:

- signalling connection, running the designated signalling protocol;
- connection for information transfer, optionally running communication protocols.

For signalling connections, Profile A provides functionality to set-up and breakdown connections. The functionality is covered by one message access at the top of the layer 3 of the signalling protocol.

If the PUF has created a NCO associated with external equipment, Profile A provides functionality to set-up and breakdown connections and all user actions with the external equipment (on-hook, off-hook, dialling) are taken into account by the signalling part. Furthermore, some external equipments are able to manage ISDN signalling when the host is off.

In case of telephony, additional functionality can be available, which allows the temporary breakdown (suspend) and subsequent re-establishment of connections (resume).

As a NCO is coupled to a single PUF, connection passing between PUFs cannot be accommodated.

5.2.1.3.3.2 Support of supplementary services

Supplementary services, as provided by the connection management of ISDN, are available to the PUF when PUF co-ordination case applies. The PUF is responsible for the handling of the connection management and can, therefore, control the supplementary services provided via the signalling.

To facilitate the use of certain supplementary services described in the ISDN MOU of European Community (1989), a special way of coding has been introduced in Profile A.

In general, supplementary services provided by ISDN may be provided by the use of transparent coding of the supplementary services. The transparent coding shall be provided as an optional feature.

NOTE: The use of supplementary services, when the co-ordination function is in use by the NAF is for further study.

5.2.1.3.4 Data management

The data management functionality covers two aspects:

- establish data connections on already established physical connections;
- exchange data.

The User Plane of Profile A provides the functionality defined by the data management.

For user data transfer, Profile A provides access to various User Plane protocols running in the ISDN network layer. Depending on the selected User Plane protocol the User Plane provides access to a network layer (Layer 3), link layer (Layer 2) or transparent (Layer 1) connection.

Selection of the User Plane protocol can be achieved using the resource management functionalities (Creation or Modification of an NCO).

For every type of connection it is important that there exists a signalling connection before any data access can be done. In general, unless a PUF makes use of the co-ordination function provided by the NAF, the establishment of the signalling connection is achieved by using the Control Plane functionality, whereas the establishment of data access is achieved by use of the User Plane functionality.

When using a connection with a transparent User Plane protocol and a NCO which is associated with external equipment, the data generated on the connection shall be sent to the external equipment rather than used to generate User Plane messages.

A detailed description of the available protocols and of the corresponding User Plane messages, sequencing and parameters can be found in subclause 5.6.

5.2.1.4 Relating functionality to planes

When relating functionality to the planes, the following relations apply:

- the Administration Plane of this profile provides the functionality defined by the resource management;

- the Control Plane of this profile provides the functionality defined by the connection management;

- the User Plane of this profile provides the functionality defined by the data management.

Inside the planes the functionality is described using operations or operational groups.

5.2.1.4.1 Optional features

When relating functionality to the planes, there shall be some operations or operational groups which are not mandatory for a NAF to supply.

In the description of the planes there are indications of which operations or operation groups are mandatory or optional.

The fact that the description allows optional features for the NAF does not mean that Profile A contains any optional features. Profile A, as an interface specification, shall allow the exchange of any message. Optional here refers to the availability of these features to the PUF, supplied by the NAF. If the PUF requests a feature which is not provided by the NAF, the PUF shall be informed of this.

5.2.1.4.2 Administration Plane

The Administration Plane provides access to operations which facilitate management of connections like definition and management of attribute and address sets as well as management of network connection objects. Furthermore, the following miscellaneous operations are provided via this plane:

- error report operation;
- security operation;
- manufacturer specific operation;
- change of protocol during a connection.

Table 2 provides an overview of Administration Plane operations.

Table 2: Administration Plane operations

Operation name	Purpose of operation
Create NCO	Create a network connection object
Destroy NCO	Destroy a network connection object
GetInfo NCO	Obtain information about a network connection object
Error	Report non-connection related error condition
Security (note)	Manipulate security
Manufacturer Specific (note)	Request manufacturer specific functionality
Change of protocol (note)	Change the User Plane protocol on the established B-channel
NOTE: These operational groups are optional for the NAF.	

5.2.1.4.3 Control Plane

The Control Plane provides access to operations which handle the basic call control of the ISDN signalling.

In the Control Plane, there exists no clear separation of operations as in the Administration Plane. It shall be possible to distinguish between a number of operational groups in the Control Plane. Table 3 provides an overview on Control Plane operational groups.

Table 3: Control Plane operations

Operational group name	Purpose of operational group
Connection establishment	Handling incoming and outgoing calls
Connection breakdown	Handling the removal of connections or refusal of calls
User-to-user information transfer (note)	Exchanging user-to-user information and providing control for this exchange
Adjournment of calls (note)	Provision of suspending and resuming calls
Facility invocation (note)	Handling the invocation of facilities
External equipment (note)	Indicate status or change of state of external equipment
Additional information (note)	Provide a access to additional information during a call
DTMF (note)	Exchange of DTMF in band information during a call
NOTE: These operational groups are optional for the NAF.	

5.2.1.4.4 User Plane

The User Plane provides operations which facilitate establishment, data exchange and release of logical communication channels. It uses standardized services and procedures as defined for the selected user message access. A detailed description of the operations available for the various possible access (transparent, link layer, network layer) can be found in the subclause 5.6.

5.2.1.5 PUF NAF interactions

This subclause describes the type of functions which are available to the PUF in its interactions with a NAF and in which order they can be used.

For all functions the following properties apply:

- initiated by the PUF, which means that only the PUF can initiate the association the PUF to the NAF;

- requested by using function calls from the PUF to the NAF;

- performed in a synchronous manner. A PUF which requests a NAF to perform a function shall regain control of the CPU from the NAF after completion of the function call.

In the interaction between PUF and NAF the following phases can be distinguished:

- Registration Phase

  Before a PUF and a NAF can interchange information, the PUF associates with the NAF. As it is possible within a system that more than one NAF may be available, and, additionally, these may be from different NAF manufacturers, a method is defined which allows the PUF to discover which NAFs are accessible within a system. This phase is called the Registration Phase. This phase allows access to a list of accessible NAFs via the PCI-Handles. Then the PUF may discover properties of the NAF that have been selected by the PCI-Handle and establish an association to the NAF.

- Conversation Phase

  At this point PUF and NAF can exchange messages. This phase is called the Conversation Phase. The PUF controls the exchange of messages between the NAF and itself. This means that the PUF fills the message with relevant parameters and sends it to the NAF for processing, or the PUF asks the NAF to receive a message by providing resources to the NAF.
  There are two methods for a PUF to discover that the NAF has a message for it. The simplest way for the PUF to get available messages is to poll the NAF. The second method provides a mechanism to give the NAF a fast way to notify a PUF that a message is available. With this method the PUF explicitly allows the NAF to notify it on the availability of a message. This method has the advantage of introducing an efficient way of operating, for PUFs which are concerned with performance.

  For example, this method shall help PUFs which have bound to multiple NAFs. However, PUFs who use this method are more complex in design than those that do not.

- De-registration Phase

  When a PUF does not need to exchange messages with a NAF, it disassociates from the NAF. This phase is called the Deregistration Phase. This phase is important in terms of resource management in the NAF, especially for memory resources. The PUF shall disassociate to guarantee an efficient use of the resources of the global system.

Table 4 gives the list of functions grouped into their respective phases.

Table 4: Profile A functions grouped into phases

Phase	Function	Purpose of function
Registration	PciGetHandles	Provide a list of accessible NAFs and obtain their PCI-Handles
	PciGetProperty	Provide detailed information on a NAF
	PciRegister	Associate the PUF to the NAF
Conversation	PciPutMessage	Transfer a message from the PUF to the NAF
	PciGetMessage	Ask the NAF to receive a message, by providing resources
	PciSetSignal	Establish mechanism to allow the NAF to notify the PUF when a message is available
Deregistration	PciDeregister	Disassociate the PUF from the NAF

The functions shall be used in a certain order. Figure 3 presents a state diagram for the Profile A function calls.

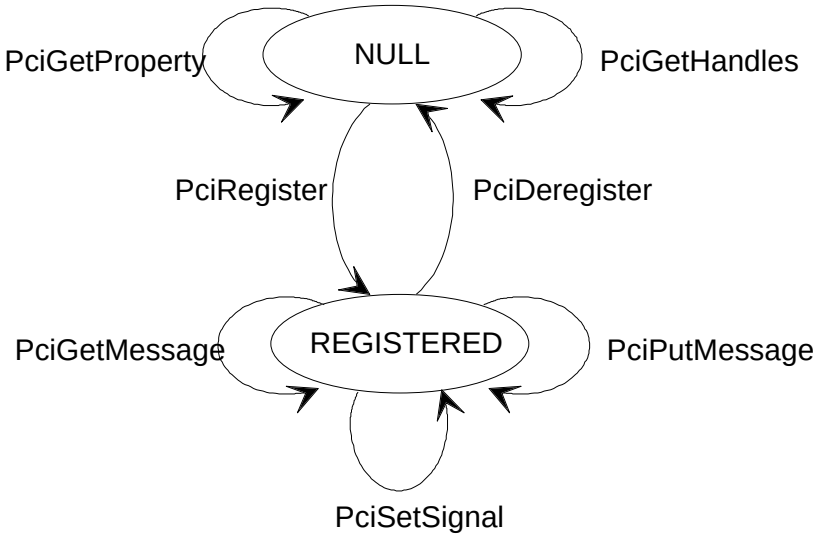


Figure 3: Profile A function calls order

The messages are transferred between the PUF and the NAF by use of the PciPutMessage and PciGetMessage functions.

5.2.1.6 Total interaction overview

As an example of the sequencing of operations, figures 4 and 5 present a chronological interaction overview of the actions the PUF shall perform to get a connection.

In these figures, the following conventions are used:

- for the complete figure, dashed lines mean optional;

- in the part of the figure on the Conversation Phase, arrows from the PUF to the NAF mean usage of PciPutMessage and arrows from the NAF to the PUF mean usage of PciGetMessage;

- text written in small letters refers to messages as described in subclauses 5.4, 5.5 and 5.6 of this ETS.

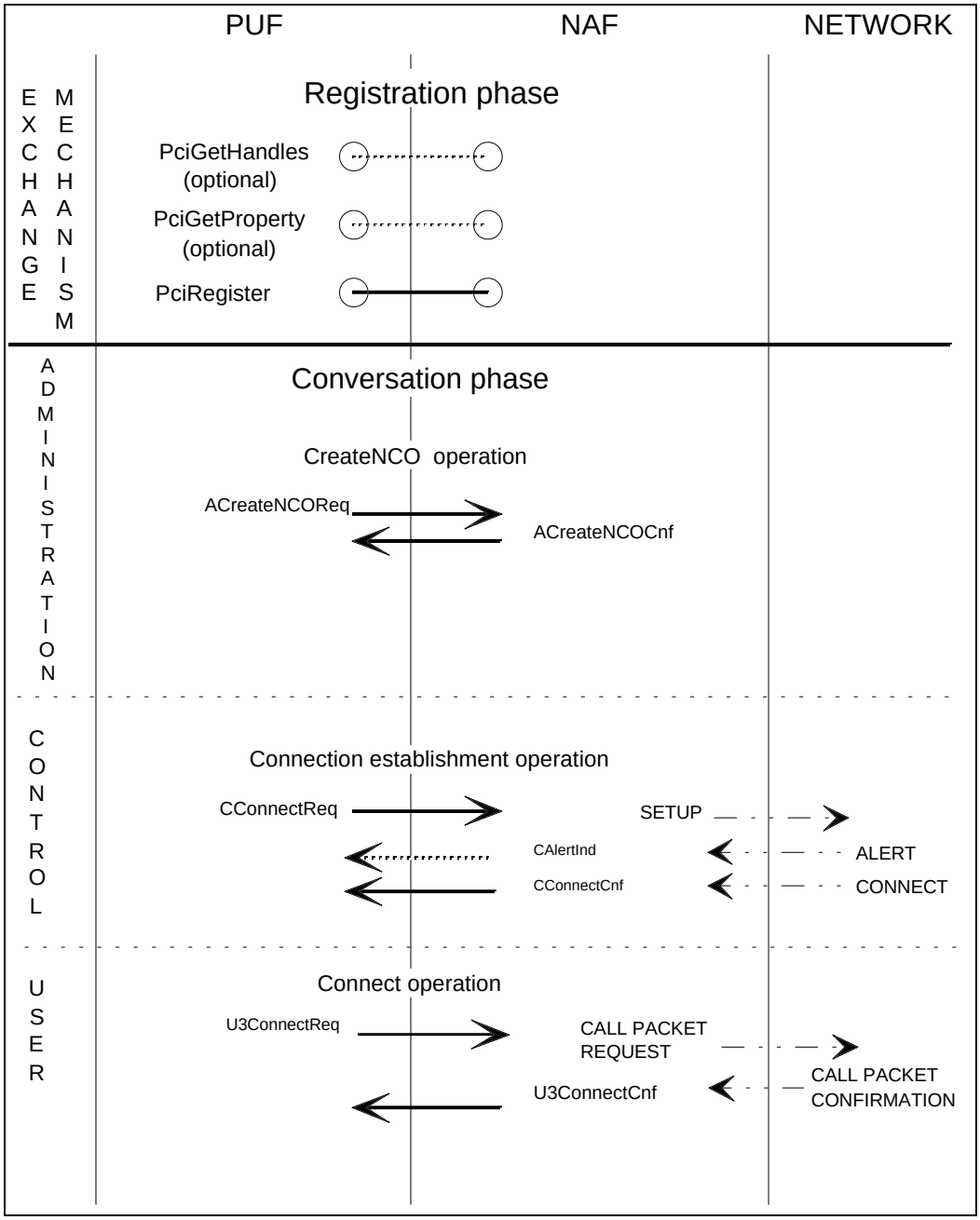


Figure 4: Sample of sequencing operations - PUF co-ordination case

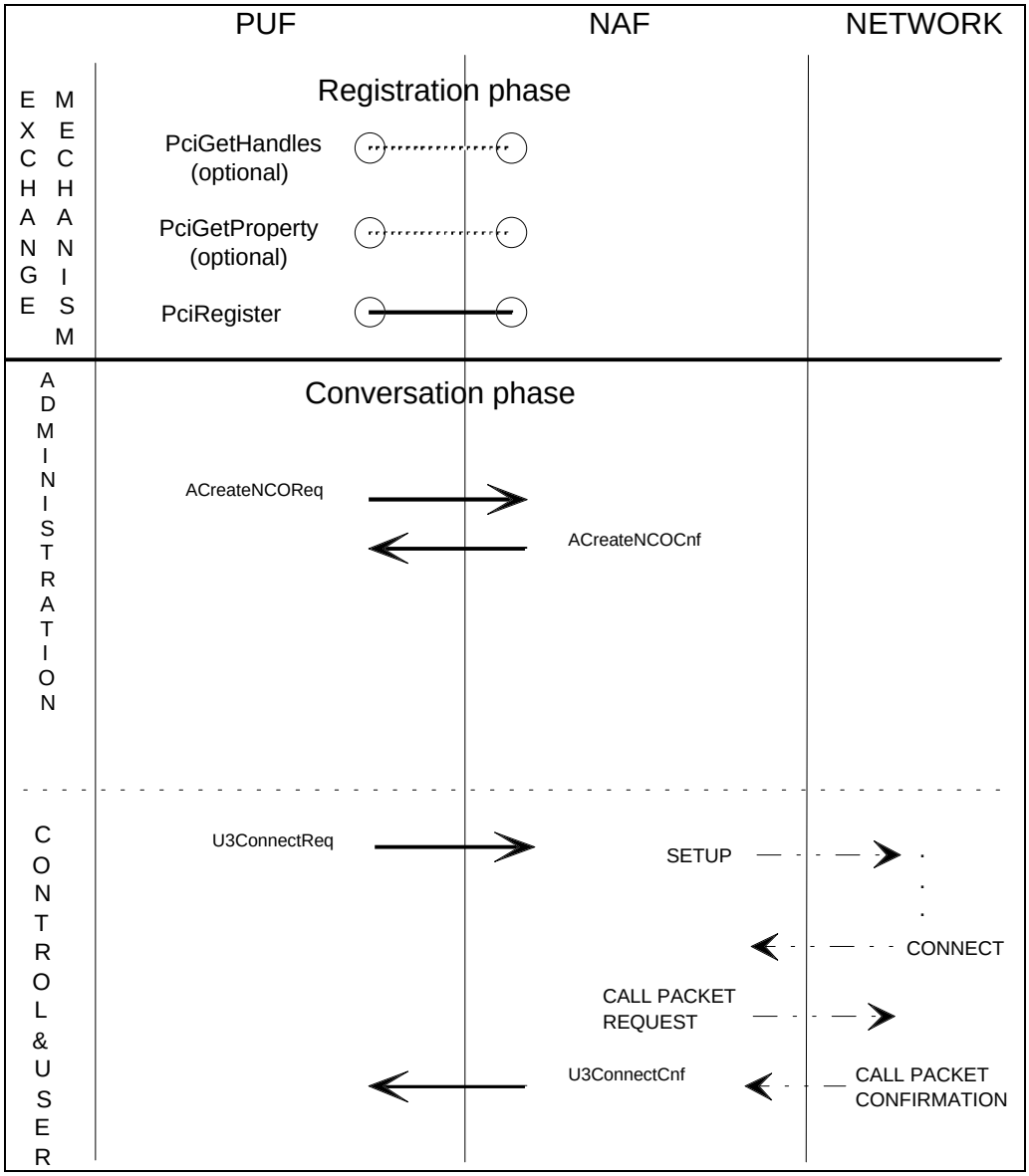


Figure 5: Sample of sequencing operations - NAF co-ordination case

5.2.1.7 Identifiers

For its operations, Profile A defines identifiers. These identifiers shall be used by the PUF to identify concrete objects or connections in an abstract manner.

This subclause summarises the identifiers used in the Profile A specification. Only the functional description of these identifiers are given. The details are introduced in later clauses.

For details of identifiers see subclause 6.5 related to the exchange mechanism.

PCI-Handle This identifier is an abstract reference to a NAF. The PCI-Handle shall be used to optionally find out information on the NAF and to register to the NAF from the exchange mechanism function **PciRegister**.

ExID This identifier is the representation of the association between a PUF and a NAF. It is provided by the exchange mechanism function **PciRegister**. It is needed in every Profile A function call in relation to this association.

Identifiers relate to the Conversation Phase of the communication between PUF and NAF. For details of these identifiers refer to subclause 5.4.

CAttributeName This identifier relates to a static attribute set of Control Plane parameters. It is represented as a name. This identifier shall subsequently be used in creating a NCO. The complete list of static attribute sets is defined in annex H.

UAttributeName This identifier relates to an attribute set of User Plane parameters. It is represented as a name. This identifier shall subsequently be used in creating a NCO. The complete list of static attribute sets is defined in annex H.

ExtEquipName This identifier relates to external equipment. It is represented as a name. This name may be obtained either implicitly or by use of the PciGetProperty function.

NCOID This identifier relates to the connection which is referred to by the PUF. It is the way for the PUF to indicate to the NAF which connection is referred to since a connection always corresponds with a NCO, this identifier is a reference to this NCO. The NCOID is provided in response to the Create NCO message.

GroupID Abstract identifier for grouping NCOs. It is User Plane protocol dependent.

RequestID This reference identifies the message which is exchanged between the PUF and the NAF in the Administration Plane. Subsequent responses to this message shall contain the same RequestID to identify the original message. It is allowed multiple asynchronous transmissions to this plane.

SelectorID This reference identifies the NCO related to the message in case of multiple NCOs matching, on an incoming call. The NAF will select only one NCO in this abstract PUF set, indicated by the same SelectorID value. This is a way for the PUF to limit the amount of NCOs selected by a NAF and then to limit the number of messages generated by the NAF in case of an incoming call.

ControllerID This identificator attaches a particular controller to an NCO or to a connection. A *ControllerID* is a number based 0. The way a NAF assigns a *ControllerID* to an adapter is NAF dependant. For incoming call, the NAF provides this *ControllerID* value if multiple adapters are installed. In case of outgoing call, a PUF is able to request an adapter during the NCO creation operation or at the call establishment. Otherwise the NAF will apply its internal rules to make the selection. If only one adapter is installed, the ControllerID is ignored. The number of installed adapters is a Property parameter.

5.2.1.8 Error handling

5.2.1.8.1 Overview

Error information is returned to the PUF by the means of function return codes and information present within messages. Generally, the function return codes provide error information generated by the passing of parameters to the NAF from the PUF and the checking of the data in those parameters. Messages contain error information reflecting the checking of the data referenced by the parameters, the processing of earlier messages or events from the protocols in use.

5.2.1.8.2 Function error handling

For each function the supplied parameter values are checked, if any of them are found to be in error then the fact shall be reported as a function return code and the action requested by the function shall not take place.

The parameter examination that takes place (and order of checking) when a function is invoked by a PUF depends on the function being invoked.

5.2.1.8.3 Message error handling

Error detection takes place at 2 stages during the processing of a Profile A message:

1) when the message is initially examined by the NAF, to ensure that it is suitable for further processing. This checking is administrative in nature, so any errors encountered are returned in Administration Plane messages;

2) when the message is processed by the NAF, the way in which error information is passed to the PUF depends on the plane that the message belongs to and the protocol underlying that plane.

The initial examination that takes place (and order of checking) when the message is first received from a PUF is as follows:

a) NAF availability is checked;
b) Message identifier is checked;
   - unknown message, not defined by Profile A;
   - unsupported message, defined by Profile A but not supported by NAF.

In the case of Administration Plane messages, any error information is returned on the corresponding confirm message. In the case of Control and User Plane messages, error information is returned in the Administration Plane AErrorInd message.

The error detection that takes place (and order of checking) when the message is processed by the NAF is protocol dependent. Error information is returned by a mechanism particular to the protocol in use. These mechanisms are described in subclause 5.9.

5.2.2 Information encoding

In subclauses 5.6 and 5.7, the types used shall be understood as:

- Octet — referred to a byte (8 bits);
- Boolean — referred to an octet with limited set of values (0 = FALSE, else = TRUE);
- Octet-string — referred to an array of octets with a variable or fixed size;
- IA5-string — referred to an Octet-string composed with octets in the IA5 alphabet.

Every parameter is encoded using Type Length Value (TLV) coding as following:

- type = 1 Octet;
- length = 1 Octet;
- value with octet boundary.

Fields included in the parameter are coded as structured information. The order of this structured information is defined by the order of the parameter itself in subclause 5.7. Omitted fields reduce the size of the parameter.

Values in parenthesis are decimal.

EXAMPLE: This example describes parameters of a CDisconnectInd message coming from the NAF.

The *CDisconnectInd* message identifier is provided by the NAF in the *MessageID* field of the PCIMPB structure as describe in the subclause 5.3. Parameters are placed into fill the *Message* parameter of the PciGetMessage function (see subclause 5.3.3.6). The *MessageActualUsedSize* field of the PCIMPB structure is filled with the 12 value.

The cause content is the "Normal call clearing" #16 cause, the diagnostic optional parameter is not provided. The NCOID value is 3. Values are given using the decimal format.

Total parameter length	12	
NCOID type	49	
NCOID length	4	
NCOID value	3	(03 00 00 00)
Cause type	15	(CauseToPUF)
Cause length	4	
Cause value	16	
Cause Standard	1	(ITU)
Cause location	1	(user)
Cause Recommendation	1	(Q.931)

In a byte oriented representation (in decimal), the content of the *buffer* results as follow:

49, 04, 03, 00, 00, 00,	(NCOID)
15, 4, 16, 01, 01, 01	(CauseToPUF)

5.2.3 Conventions

As described in subclause 5.2.1.5 "PUF NAF Interactions", the exchange of messages is realised through 2 functions, PciPutMessage and PciGetMessage, which may be called as soon as the PUF is bound to the NAF. Due to the nature of these functions, which may be used independently of each other, correlation of "get messages" to "put messages" shall be performed by the PUF. For this reason, the messages of each plane contain identifiers allowing correlation between messages.

The following subclauses describe the messages provided by each plane of Profile A, as well as the parameters used in conjunction with each message. The actual information presentation and coding for the operations and parameters is described in subclause 5.2.2, "Information encoding".

The description of the messages, their parameters and fields is independent of hardware and operating systems.

5.2.3.1 Address conventions

When using any address in this ETS, the following conventions apply:

- The called address refers to the address the sender desires to be connected to.
- The calling address refers to the local address of the sender.

5.2.3.2 Provision of information

The provision of, or requirement for, items in the message can vary. The following conventions and abbreviations are used:

M = (Mandatory): this item shall be supplied.

C = (Conditional): a condition determines if this item is supplied. The condition is explained as a comment to the item.

O = (Optional): this item may or may not be supplied. For the exchange from PUF to NAF this implies that the PUF is free to provide the item or not. For the exchange from NAF to PUF this implies that the NAF shall only supply the item if it is available.

Information coming from the NAF reflects information provided by the network.

5.2.3.3 Message conventions

This subclause presents conventions used in the tables for describing the messages.

Each message belongs to a class. With each message the class is indicated. Not all classes are available for a NAF to support. A NAF provider may choose to implement only certain classes. Each plane contains its own classes.

For each plane, a PUF can only rely on the availability of messages from class 1 (basic class). The other messages belong to additional classes. If a NAF implements an additional class, all messages of the same plane in this class shall be provided.

The message indicates its direction of transfer in the suffix part of its name. Messages with the suffix Req or Rsp are transferred from the PUF to the NAF. Messages with the suffix Ind and Cnf are transferred from the NAF to the PUF. Message identifiers are provided in decimal.

5.2.3.4 Parameter conventions

With the description of parameters the following conventions are used:

- the name of the field shall be given;

- the type of the field shall be given in decimal;

- the entity in charge of providing the content of the field - the Direction column. The following abbreviations are used:

  P in charge of the PUF;
  N in charge of the NAF;
  B both PUF and NAF can provide the content;

- the length of the field may be given. This indicates the number of octets this field shall occupy. The term octet does not refer to any hardware or operating system dependent implementation. It refers to the basic information unit in all systems.

5.2.3.4.1 Parameter ordering

No ordering between parameters in messages is needed. The ordering of the parameters is not described by the ordering in the tables.

The ordering of the fields within the parameters is defined in the subclause 5.2.2.

5.2.3.4.2 Parameter repetition

Parameters in a message can be repeated. The number of repetitions is fixed by the network or the user protocol used. If the numbers of repetition exceeds the number of repetitions allowed, remaining repetitions shall be ignored.

5.2.3.4.3 Parameter checking

No particular checking process should be performed by the NAF for parameters coming from the network.

5.2.3.5 Default philosophy

For the values of parameters and fields in parameters a default philosophy applies. This means that, if appropriate, the value "default" is shown in the description. After this value the value implied by the default is given.

The default value shall only be used in the message exchange from PUF to NAF. If a parameter is not provided in a message, the value provided during the NCO creation operation takes place.

In the exchange from NAF to PUF only the real value shall be given.

5.2.4 User Plane particularities

The User Plane messages provide access to different User Plane protocol stacks. Depending on the selected User Plane protocol the User Plane provides access to a network layer (Layer 3), link layer (Layer 2) or transparent (Layer 1) connection. A detailed description of the available protocols and of the corresponding User Plane messages, sequencing and parameters can be found in subclause 5.6.

The subclause 5.7 describes parameters for messages of the Administration and the Control Plane. Parameters for messages of the User Plane are described in subclause 5.6, on a protocol basis.

5.3 Exchange method

This subclause describes the exchange method and the exchange functions, which are used to achieve the local exchange of information between a PUF and a NAF. Since the implementation of the exchange functions is operating system dependent, they are described in a generic way. The exchange method for a real environment is contained in clause 7, for various operating systems. They include the binary representation, some code samples in Programming Language C and all information attached to a particular operating system.

Since the NAF side implementation of the exchange functions depends on the underlying operating system, the PUF code calling this function is operating system dependent as well. To be source code portable between different operating systems, the PUF should encapsulate the code calling the NAF by a functional interface, which resembles the generic exchange functions described in this subclause.

The exchange functions pass and return parameter values. These values are based on the generic types shown in table 5.

Table 5: Generic types of exchange method

Generic Type	Explanation
PCI_INTEGER	Binary represented signed integer value, covering in the minimum the range of $-2^{15}+1$ .. $+2^{15}$
PCI_BYTEARRAY	Array of binary represented byte values, used to present characters. The sign extension on the byte value is undefined. No arithmetic shall be performed on it.
PCI_EXID	Implementation dependent type for presenting the PCI Exchange-ID.
PCI_HANDLE	Operating system dependent type for presenting the PCI-Handle information.
PCI_PROCEDURE	Operating system dependent type for presentation of procedure addresses.

Dependent of the operating system, the parameters are passed either by value or by reference. The way parameters are passed is defined in the relevant subclause of clause 7.

General conventions for this generic presentation:

- the function name is prefixed by the letters "Pci";
- each function returns a completion code. Any other value than Success (0) for the completion code indicates an error.

5.3.1 Registration phase

5.3.1.1 Overview

Before a PUF and a NAF can interchange information the PUF shall associate with the NAF. For this association the PUF shall specify the PCI-Handle of the NAF it wants to associate with.

To support many NAF implementations, possibly from different manufacturers, a method is defined which allows the PUF to discover which NAFs are accessible from within a system. For this the optional[1)] function **PciGetHandles** allows the PUF to get a list of accessible PCI-Handles. Subsequently, the PUF can extract a PCI-Handle from the list. The presentation of PCI-Handle is described in the operating system specific documentation.

If used the **PciGetHandles** function should be the very first function called by a PUF since it makes all PCI-Handles available. The interworking with other exchange functions is shown in figure 3.

---

1) The use of this function is optional for the PUF, but it's implementation (provision) is mandatory for the NAF.

Another optional[1)] function available in the registration phase is the **PciGetProperty** function. It allows the PUF to learn the properties of the NAF. On call the PUF has to give the PCI-Handle of the NAF of interest. As a result the PUF obtains a list of the static properties of the NAF.

Since the obtained properties contain information about special NAF features, the PUF can use this information to select the NAF(s) it wants to register with. Examples of these special features are a handset or security features.

The only non-optional function of the registration phase is the **PciRegister** function. It allows the PUF to associate with the NAF. The PUF shall provide the PCI-Handle of the NAF it wants to associate with. As a result, an identifier for the association between the PUF and the NAF will become available. This identifier shall be given in subsequent exchange function calls of this association during the conversation and deregistration phase.

The following terms are used in conjunction with the registration phase:

NAF-Property: Structured information describing the characteristics (properties) of a NAF. The NAF-Property is implemented system independent using TLV coding (see clause 6.3). Hence it shall be encoded using the same algorithm as used for encoding of messages. In a multiple NAF environment, a PUF can use this information to select a specific NAF.

PCI-Handle: NAF access information. This information shall be supplied to the functions of the registration phase in order to find and access a NAF. Implementation of the PCI-Handle is operating system dependent. For example the PCI-Handle might be a name, a file-path or a function address.

NOTE: In many NAF environments a PUF may use the optional functions PciGetHandles and PciGetProperty as described above to select the NAF which best suits it's needs. For more details on the **PciGetProperty** function refer to subclause 5.3.1.3.

5.3.1.2 PciGetHandles

This function allows a PUF to ask how many NAFs are accessible and to obtain their PCI-Handles. Using the PCI-Handle, the PUF can subsequently get the NAF-Property or register with this NAF.

Function Name: PciGetHandles

Function Return Value: Errorcode (PCI_INTEGER)

Success
QueryEntityNotAvailable
InvalidHandlesBuffer
HandlesBufferTooSmall

Parameters

Name	Generic Type	Call or Return Value	Comment
MaxHandles	PCI_INTEGER	Call Value	Maximum number of PCI-Handles that can be received
PCIHandles	Array of PCI_HANDLE	Call Value	a buffer, big enough to receive the requested maximum amount (MaxHandles) of PCI-Handles.
ActualHandles	PCI_INTEGER	Return Value	Actual number of PCI-Handles returned in the given buffer.

The PUF shall give a buffer and its size to get the list of available PCI-Handles.

The PUF receives the actual number of PCI-Handles copied into the buffer. If this number is greater than the size of the buffer allows, the buffer is not filled and the HandlesBufferTooSmall error is returned. In this case, the PUF shall provide another larger buffer to get the complete list of PCI-Handles.

5.3.1.3 PciGetProperty

This function allows a PUF to obtain the NAF-Property. The PUF shall supply a PCI-Handle as a call value. A PUF can obtain a PCI-Handle either by the use of the optional **PciGetHandles** function or by use of other means (e.g. local knowledge).

Function Name: PciGetProperty

Function Return Value: Errorcode (PCI_INTEGER)

Success
InvalidPCIHandle
NAFnotAvailable
NAFBusy
PropertyBufferTooSmall

Parameters

Name	Generic Type	Call or Return Value	Comment
PCIHandle	PCI_HANDLE	Call Value	PCI-Handle presentation and values are operating system dependent.
MaximumSize	PCI_INTEGER	Call Value	Maximum size of property allowed on return.
NAFProperty	PCI_BYTEARRAY	Return Value	Property returned. Property is TLV coded, hence not system dependent (see table 6).
ActualSize	PCI_INTEGER	Return Value	Actual size of property returned.

The parameters of NAF-Property are shown in table 6.

Table 6: TLV coded NAF-Property parameter

Parameter	Provided	TLV Coding (see note)			Comment and values
		TypeID	Length	Value	
Product	M	1	1..32	Octet	Octet string indicating NAF Product
Manufacturer	M	2	1..32	Octet	Octet string indicating NAF Manufacturer.
AccessClass	M	3	1	Octet	Basic Rate (1) or Primary Rate (2)
UserProtocolL3*	M	4	1..4	Octet	Give the supported layer 3 protocols. May be a NAF selection criteria. For defined value see subclause 5.7.48.
UserProtocolL2*	M	5	1..2	Octet	Give the supported layer 2 protocols. May be a NAF selection criteria. For defined value see subclause 5.7.48.
BChannels	M	6	1	Octet	Number of B-Channels
BPermanent	O	7	1	Octet	Number of Permanent B-Channels
DPermanent	O	8	1	Octet	Number of Permanent D-Channels
AplaneClass*	O	9	1	Octet	Additional Administration Plane functions supported. identified by class. Valid value: 2..4
CplaneClass*	O	10	1	Octet	Additional Control Plane functions supported identified by class: Valid values are in range 2..7.
SuppService*	O	11	1..16	Octet	Specification of supplementary services. See table 7 below for precoded supplementary services.
ExtEquipName*	O	12	2..17	Octet	In order, type and name of external equipment. See subclause 5.7.31.
AdditionalUserProtocol*	O	13	1..16	Octet	Additional User Plane protocols. For further extension.
PCIVersion*	O	15	1	Octet	PCI version supported
ControllerNumber	O	20	1	Octet	Number of controllers available. Default is 1.
MailBoxMan	O	21	1	Octet	Mail box management. No (0) Yes (1)
Password	O	22	1	Octet	Password management. No (0) Yes (1)
Timer	O	23	1	Octet	NAF has timer. No (0) Yes (1)
StripControl	O	24	1	Octet	bit 0: On transmission bit 1: On reception bit 2: Data/Time display bit 3: Page numbering bit 4: ID display

(continued)

Table 6 (concluded): TLV coded NAF-Property parameter

Parameter	Provided	TLV Coding (see Note)			Comment and values
MailBoxNum	O	25	1..2	Octet	Maximum number of mail boxes memorised
PollingCap	O	26	1	Octet	Polling capacity: bit 0: on transmission bit 1: on reception
TransScanLine	O	27	1	Octet	Minimum transmission time for scan line: (0) 0 ms (1) 5 ms (2) 10 ms (3) 20 ms (4) 40 ms
DataRate	O	28	1	Octet	Maximum data signalling rate in message mode: (1) 2 400 bits/s (2) 4 800 bits/s (3) 7 200 bits/s (4) 9 600 bits/s (5) 12 000 bits/s (6) 14 400 bits/s (7) 64 kbits/s
ECMMan	O	29	1	Octet	ECM management: No (0) Yes (1)
CodingType	O	30	1	Octet	Coding format supported: (0) T4 (1) T4 bi-dimensional (2) T6 (3) BTM (Basic Transfer Mode) (4) DTM (Document Transfer Mode) (5) BFT (Binary File Transfer) (6) EDI (Edifact Transfer) (7) Character mode (8) Mixed mode
Measure	O	31	1	Octet	Measuring units supported: (0) Metric based (1) Inch based
PageHeight	O	32	1	Octet	Page height supported: bit 0: A4 bit 1: B4 bit 2: Unlimited
PageWidth	O	33	1	Octet	Page width supported: bit 0: 1 728 pels bit 1: 2 048 pels bit 2: 2 432 pels
Resolution	O (*)	34	1	Octet	Resolutions supported: (0) R8 x 3,85 lines/mm (98 dpi) (1) R8 x 7,7 lines/mm (196 dpi) (2) 200 x 200 dpi (3) 300 x 300 dpi (4) 400 x 400 dpi (5) R8 x 15,4 lines/mm (392 dpi) (6) R16 x 15,4 lines/mm (392 dpi)

NOTE: There may be other TypeID values defined in other clauses.

* Parameter may be repeated.

Table 7: Precoded supplementary services

Identifier	Supplementary services
'AOC-D'	Advise of charge during call.
'AOC-E'	Advise of charge at the end of call.

5.3.1.4 PciRegister

This function allows a PUF to be associated to a NAF.

As a calling parameter, the PUF provides the PCI-Handle of the NAF it wants to register with. In addition two structures are passed on the function stack:

- the PCIRegisterInfo structure and
- the PCIOpSysInfo structure.

The PCIRegisterInfo structure contains PUF and NAF specific parameters which shall be passed between the two entities to ensure proper co-operation. The PCIRegisterInfo structure is shown in table 8 below.

The PCIOpSysInfo structure contains operating system dependent information to be exchanged between PUF and NAF.

Table 8: Structure of the PCIRegisterInfo structure

Structure Field	Generic Type	Call or Return Value	Explanation
PUFVersion	PCI_INTEGER	Call Value	The version of the Profile A the PUF wants to use. Can be set to 0 in any case (Default).
PUFType	PCI_INTEGER	Call Value	The type of PUF. This parameter is for future extensions (e.g. allow specific type of PUFs like network management PUFs) Currently this value shall be set to 0!
MaxMsgSize	PCI_INTEGER	Return Value	Maximum size of a message the NAF guarantees to deal with: NAF will neither deliver messages bigger in size nor does it guarantee to accept bigger ones from PUF!
NOTE: The PUFVersion number which equals the major revision number is defined as version 2. The PUFType value is for further extensions and is currently assigned to 0.			

As a return value the Exchange Identifier (ExID) becomes available, which identifies the exchange link between PUF and NAF. The ExID shall be provided to other functions of the exchange method during the conversation and the deregistration phase.

Function Name: PciRegister

Function Return Value: Errorcode (PCI_INTEGER)

Success
InvalidPCIHandle
NAFnotAvailable
NAFBusy
MaxPUFsExceeded
InvalidPUFType
InvalidPUFVersion
InvalidRegisterInfoStructure
InvalidOpSysInfoStructure[2)]

---

2) For more (operating system specific) error codes refer to the subclause 5.9.5.

Parameters

Name	Generic Type	Call or Return Value	Comment
PCIHandle	PCI_HANDLE	Call Value	PCI-Handle presentation and values are operating system dependent.
PCIRegisterInfo	PCIRegisterInfo structure	Call Value	Contains PUF-NAF interaction specific information like PUFVersion and PUFType (see table 8)
PCIOpSysInfo	PCIOpSysInfo structure	Call Value	Contains Operating system dependent information. For details refer to clause 7 and annex J.
ExID	PCI_EXID	Return Value	Exchange-ID.

5.3.2 Deregistration phase

This phase is the last phase in the information exchange between PUF and NAF. When a PUF wants to disassociate from the NAF it shall invoke the **PciDeregister** function. The use of the **PciDeregister** function is mandatory prior to PUF termination.

When the PUF disassociates using this function the NAF frees any resources allocated for this PUF, such as clearing already existing connections.

5.3.2.1 PciDeregister

This function disassociates a PUF from a NAF. The association between the PUF and NAF is identified by the ExID.

Function Name: PciDeregister

Function Return Value: Errorcode (PCI_INTEGER)

Success
InvalidExID
NAFnotAvalaible
NAFBusy

Parameters

Name	Generic Type	Call or Return Value	Comment
ExID	PCI_EXID	Call Value	Exchange-ID as received as result of previous PciRegister function.

On return the ExID used becomes invalid, even if the error code returned indicates an error during deregistering. No further access to the NAFcan take place using this ExID.

5.3.3 Conversation phase

In the conversation phase the interaction between the PUF and the NAF consists of message and data exchange. This exchange is carried out by the generic exchange functions **PciPutMessage** and **PciGetMessage** respectively. Messages are provided, in both directions, one by one and entirely. Message and data are associated. The PCI Message Parameter Block (PCIMPB) structure contains information on message and data pointers.

Messages are processed by the NAF in an asynchronous way, but the execution of the exchange functions is synchronous. As the PUF controls the exchange of messages, messages are transmitted or received only when the PUF wishes.

5.3.3.1 Sending messages

The PciPutMessage function is provided for the PUF to send messages to the NAF. Before using this function the PUF shall fill the PCIMPB with the appropriate values and shall provide the addresses of the message and the data buffer. The latter only in case the PUF sends data associated with the message. The PCIMPB contains the Message Identifier and details concerning the usage of the message and data buffers.

5.3.3.2 Receiving messages

To get a message the PUF simply issues a **PciGetMessage** function call. The PUF can use this function to poll for message availability. On function return it is indicated if there was a message transfer or not. To avoid polling, the PUF may chose to be informed via a *signal-like* mechanism as soon as a message is available. The NAF will inform the PUF for each event of message availability. This mode of operation improves the global performance of the system. However, in any case the PUF shall obtain the message itself via a **PciGetMessage** function call.

5.3.3.3 Receiving messages using the polling method

To receive a message using this method, the PUF shall poll the NAF repeatedly to check if a message is available. If no message is available, this will be indicated in a special way.

To be able to receive a message the PUF provides the NAF with a correctly set-up PCIMPB, which shall contain the addresses of a message and a data buffer respectively. The size of the message and data buffers have to be big enough to receive the expected message. However, provision of a data buffer is optional, as data is not provided with all messages. Local knowledge in the PUF is required to determine the necessity for this buffer. The NAF indicates the total length used for each buffer. If no data is available with the message, this will be indicated by the value zero for the length used. The absence of a message is indicated by the NOMESSAGE (0) type in the MessageID field of the PCIMPB.

5.3.3.4 Receiving messages using signal method

To receive a message using this method, first, the PUF shall establish a mechanism for the NAF to notify the PUF when a message is available. This is accomplished using the **PciSetSignal** function.

This method allows a NAF to indicate that a message is available for the PUF without waiting for the PUF to use the **PciGetMessage** function**.** The indication does not involve transfer of the message from the PUF to the NAF.

The NAF notifies the PUF each time a new message is available. It will do so until the **PciSetSignal** function is used to remove the notification mechanism.

To receive the message from the NAF, the PUF shall use the **PciGetMessage** function as described in the previous subclause.

The function calls the PUF is allowed to invoke while processing the notification may be restricted. These restrictions are dependent on the operating system.

5.3.3.5 PCI Message Parameter Block (PCIMPB)

Table 9 shows the structure of the PCI Message Parameter Block (PCIMPB).

Table 9: Structure of the PCI Message Parameter Block (PCIMPB)

Structure Field	Generic Type	Explanation
MessageID	PCI_INTEGER	Message identifier. Shall be provided by PUF on invocation of PciPutMessage, available for PUF on return of PciGetMessage.
MessageMaximumSize	PCI_INTEGER	Maximum size of message. To be provided on calls to PciGetMessage.
MessageActualUsedSize	PCI_INTEGER	Actual used size of message. Shall be provided by PUF on calls to PciPutMessage, will be available to PUF on return of PciGetMessage.
DataMaximumSize	PCI_INTEGER	Maximum size of data buffer. To be provided on calls to PciGetMessage.
DataActualUsedSize	PCI_INTEGER	Actual used size of data buffer. Shall be provided by PUF on calls to PciPutMessage, will be available to PUF on return of PciGetMessage.

Figure 6 presents how messages are sent or received.

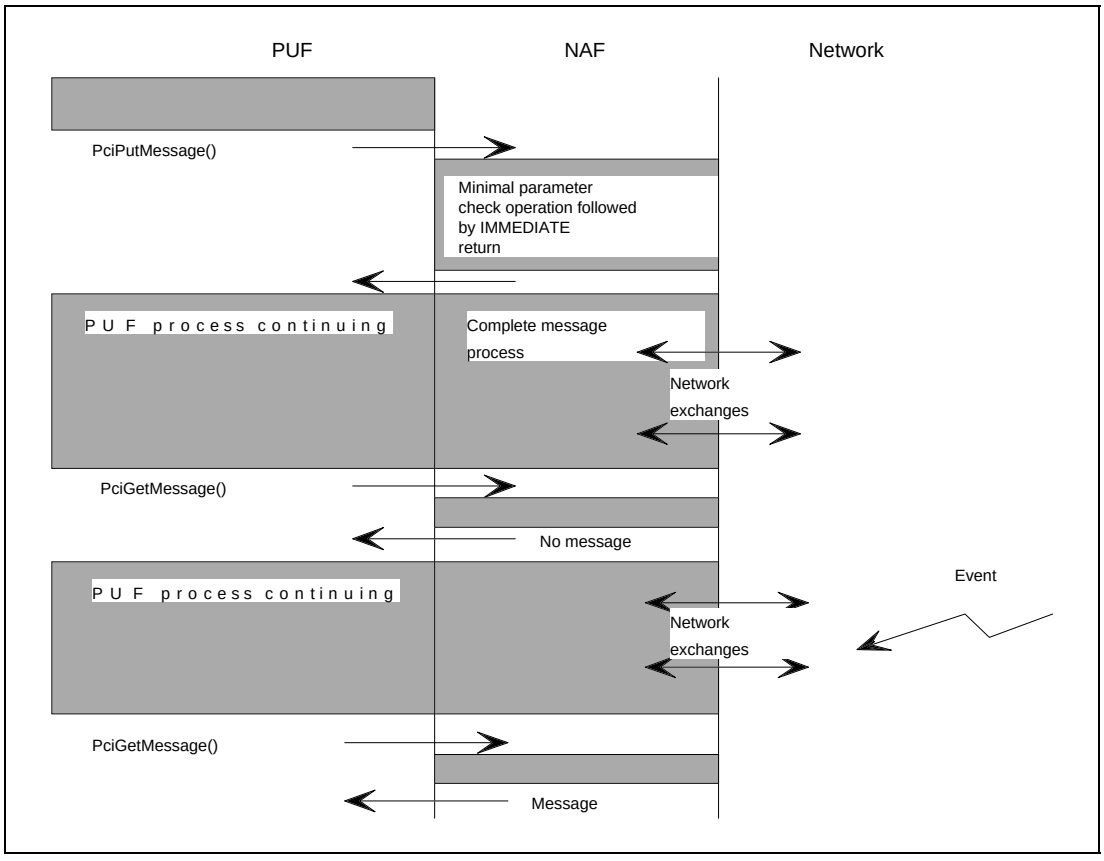


Figure 6: Process to send or to receive messages

5.3.3.6 PciPutMessage

This function allows a PUF to transmit a message to a NAF.

Function Name: PciPutMessage

Function Return Value: Errorcode (PCI_INTEGER)

Success
InvalidExID
NAFnotAvailable
NAFBusy
InvalidPCIMPB
InvalidMessageBuffer
PCIMPBTooSmall
MessageBufferTooSmall
DataBufferTooSmall
MessageTooLarge
DataBufferRequired

Parameters

Name	Generic Type	Call or Return Value	Comment
ExID	PCI_EXID	Call Value	Exchange-ID as received as result of previous PciRegister function.
PCIMPB	PCIMPB structure	Call Value	PCI Message Parameter Block.
Message	PCI_BYTEARRAY	Call Value	Message to be sent to NAF
Data	PCI_BYTEARRAY	Call Value	Data associated with the message

The PUF indicates the type of the message in the MessageID field of the PCIMPB.

The PUF indicates the size of the buffer(s) in the ActualUsedSize field of the PCIMPB for the respective buffers, i.e. MessageActualUsedSize for the message buffer and DataActualUsedSize for the data buffer.

It is allowed to provide only a message buffer without a data buffer or a data buffer without message buffer. However the PCIMPB structure is always required. To indicate absence of a buffer, the PUF may specify no buffer address instead by supplying a NULL (0) value.

5.3.3.7 PciGetMessage

This function allows a PUF to get a message from a NAF.

Function Name: PciGetMessage

Function Return Value: Error code (PCI_INTEGER)

Success
InvalidExID
NAFnotAvailable
NAFBusy
InvalidPCIMPB
InvalidMessageBuffer
PCIMPBTooSmall
MessageBufferTooSmall
DataBufferTooSmall
MessageTooLarge

Parameters

Name	Generic Type	Call or Return Value	Comment
ExID	PCI_EXID	Call Value	Exchange-ID as received as result of previous PciRegister function.
PCIMPB	PCIMPB structure	Call Value and Return Value	PCI Message Parameter Block.
Message	PCI_BYTEARRAY	Return Value	Message received from NAF.
Data	PCI_BYTEARRAY	Return Value	Data associated with the message.

The PUF is in charge of providing buffers. If a buffer is to small to receive the message or data provided by the NAF, the NAF will return an error.

The PUF indicates the maximum size of the buffer(s) in the MaximumSize fields of the PCIMPB for the respective buffers, i.e. MessageMaximumSize for the message buffer and DataMaximumSize for the data buffer

On return, the NAF will indicate the size of the buffer(s) in the ActualUsedSize field of the PCIMPB for the respective buffers used.

To indicate no message, the NAF fills the MessageID field in the PCIMPB with NOMESSAGE (0).

5.3.3.8 PciSetSignal

This function allows a PUF to ask for notification when an event occurs. An event is any incoming message from the network or from the NAF. The signal mechanism will stay in effect until the PUF disassociates from the NAF or explicitly shuts-down the notification action (see below).

Function Name: PciSetSignal

Function Return Value: Errorcode (PCI_INTEGER)

Success
InvalidExID
NAFnotAvailable
NAFBusy
InvalidSignalNumber

Parameters

Name	Generic Type	Call or Return Value	Comment
ExID	PCI_EXID	Call Value	Exchange-ID as received as result of previous PciRegister function.
Signal	PCI_INTEGER	Call Value	Value is operating system dependent.
SignalProcedure	PCI_PROCEDURE	Call Value	Value and presentation is operating system dependent.

The real signal mechanism used is operating system dependent.

Any new **PciSetSignal** call overwrites the previous one.

The signal mechanism can be stopped by supplying a NULL(0) value instead of Signal and SignalProcedure values during call.

5.4 Administration Plane messages

The Administration Plane messages are divided into the following classes:

1) management of network connection objects and error report messages;
2) management of connection security;
3) NAF manufacturer messages;
4) protocol change messages.

For management of NCOs there are messages available for creating and destroying a connection object. During creation of an NCO static or dynamic attribute and address sets are linked to the created NCO. On conclusion of the creation of a NCO, a NCOID becomes available, which shall be used in subsequent user or Control Plane operations related to the created NCO. A collection of predefined attribute sets is presented in annex H. For reporting of error information a single message is provided by the NAF. This is used to report general error conditions.

For security to be used on connections there are messages available to request security be used or stopped on a connection. These messages are optional and may not be provided by all NAFs. Their availability shall be indicated in the properties definition provided by the NAF.

To access manufacturer specific features there are messages available These messages are optional and may not be provided by all NAFs. Their availability shall be indicated in the properties definition provided by the NAF. The information exchanged between PUF and NAF is dependent on the implementation of the feature and is, therefore, not covered in this ETS.

There are messages available to request a change of the User Plane protocol associated with a NCO. These messages are optional and may not be provided by all NAFs. Their availability shall be indicated in the properties definition provided by the NAF.

All request messages of the Administration Plane may contain a request identifier (RequestID). This identifier, if assigned by a PUF on a request message, is returned by the NAF on the related confirm message.

Table 10 gives an overview of Administration Plane messages. The messages themselves are described in detail in the following subclauses.

Table 10: Administration Plane messages

Mess. Identif	Class	Message Name	Purpose of Message
101	1	ACreateNCOReq	Request to create a network connection object.
102	1	ACreateNCOCnf	Confirmation of the "CreateNCO" operation.
103	1	ADestroyNCOReq	Request to destroy a network connection object.
104	1	ADestroyNCOCnf	Confirmation of "DestroyNCO" operation.
105	1	AGetNCOInfoReq	Request information concerning a specific NCO.
106	1	AGetNCOInfoCnf	Confirmation reporting information for the relevant NCO.
108	1	AErrorInd	Indicate that a non-protocol related error has occurred.
109	2	ASecurityReq	Request to engage/stop security algorithm.
110	2	ASecurityCnf	Confirmation to engage/stop security algorithm.
111	3	AManufacturerReq	Request for a specific manufacturer functionality.
112	3	AManufacturerInd	Provide the PUF with information linked to the requested functionality.
113	4	AChangeNCOReq	Request for a change on an existing NCO.
114	4	AChangeNCOCnf	Confirmation on changing the existing NCO.

5.4.1 ACreateNCOReq

Class: 1 (Basic Class).

Description: Request message for creating a NCO.
The PUF has to provide a NCOType which identifies the type of NCO which is to be created. Depending on this type, there are more parameters needed (conditional parameters). For details refer to tables 11 and 12.
The PUF can supply a unique request identifier (RequestID) which can be used to identify the corresponding confirmation message of this operation.

Parameters:

Name	Required	Comment
RequestID	O	Request identifier, generated by the PUF.
NCOType	M	Specification of NCO type.
CDirection	C	Determines how NCO shall be used, for the Control Plane. It is absent if the NCOType value is U3 else it optional.
UDirection	C	Determines how NCO shall be used, for the User Plane. This parameter is User Plane Protocol dependent.
CAttributeName	C	Name of static Control Plane attribute.
CAttribute parameters	C	Control Plane attribute parameters. Mutually exclusive with CAttributeName; see table 64 for more details.
UAttributeName	C	Name of static User Plane attribute.
UAttribute parameters	C	User Plane attribute parameters. Mutually exclusive with UAttributeName; see the relevant User Plane protocol clauses for more details.
CAddress parameters	O	Control Plane address; see table 67 for more details.
UAddress parameters	O	User Plane address; see the relevant User Plane protocol clauses for more details..
GroupID	C	Required if NCO is to be grouped. This parameter is User Plane protocol dependent.
SelectorID	O	Helps the NAF select the right NCO.
CPMessageMask	O	ISDN message filter. If not provided, the PUF will receive any Control Plane message.
CPParameterMask	O	ISDN Control Plane parameter Filter. If not provided, the PUF will receive any Control Plane parameter.
ControllerID	O	Identify a controller. If not provided, this NCO concerns all available controllers.

Remark: See also subclause 5.8 on usage of the NCO.

Related: ACreateNCOCnf.

5.4.2 NCOType and conditional parameter specification

This subclause defines the NCOTypes usable within a ACreateNCOReq.

Currently there are four types of NCOs defined. These types are shown in table 11.

For NCOTypes supporting a User Plane access, tables 11 and 12 only show the general forms of the NCOType. A detailed description of the specific NCOType usable with a specific User Plane protocol can be found in subclause 5.6.

Table 11: NCOTypes

NCOType	NCO allows PUF ...
C	... signalling access only.
C/U	... signalling and User Plane access (PUF co-ordination functionality).
U3	... U3 User Plane access with signalling in charge to the NAF (NAF co-ordination functionality).
U3G	... User Plane access to additional virtual circuits. This NCO shall be grouped to an already created U3 or C/U type NCO.

Table 12 shows which conditional parameters shall be specified in the ACreateNCOReq message in relation to the selected NCOType.

Table 12: Specification of conditional ACreateNCOReq message parameters

NCOType	SigAttribute Type	UsrAttribute Type	SigAddress Type	UsrAddress Type	GroupID
C	C Attribute		C Address		
C/U	C Attribute	U Attribute	C Address	U Address	
U3	C Attribute	U Attribute	C Address	U Address	
U3G		U Attribute		U Address	Reference to NCO See note
NOTE:	If an NCO is to be grouped - which can only be done in case of the U3G NCOType - a reference by GroupID to an already created NCO of type U3 or C/U shall be provided. Therefore, the creation of such an NCO type shall have been carried out successfully in order to have the GroupID available.				

5.4.3 ACreateNCOCnf

Class: 1 (Basic Class).

Description: Confirmation message of the CreateNCO operation requested previously. The confirmation message can be correlated to the correct ACreateNCOReq message by use of the returned RequestID.
The confirmation message may contain the NCOID which shall be used on further requests through the User or Control Plane related to the created NCO as well as the GroupID which shall be used for subsequent ACreateNCOReq messages, if grouping to the created NCO is intended.

Parameters:

Name	Provided	Comment
RequestID	C	Provided if supplied on request message.
CompletionStatus	M	Completion status of the CreateNCO operation of the NAF.
NCOID	C	NCO identifier if CompletionStatus Success else absent.
GroupID	C	Group identifier, provided if NCO created was of type C/U or U3 and if CompletionStatus Success.

Related: ACreateNCOReq.

5.4.4 ADestroyNCOReq

Class: 1 (Basic Class).

Description: Destroys an existing NCO created by the same PUF. The PUF can supply a request identifier (RequestID) which can be used to identify the corresponding confirmation message of this operation.

Parameters:

Name	Required	Comment
RequestID	O	Request identifier, generated by the PUF.
NCOID	M	Identifier of NCO to be destroyed.

NOTE: NCO may not be destroyed if it is in use for an established connection or a connection that is attempting to be established. When a non-grouped NCO is destroyed, any NCOs grouped to it become unusable except when the grouped NCO relates to an established connection or a connection that is attempting to be established. In this case, the NCO remains usable until the related connection is removed. An unusable NCO can only be destroyed using the ADestroyNCOReq message.

Related: ADestroyNCOCnf.

5.4.5 ADestroyNCOCnf

Class: 1 (Basic Class).

Description: Confirmation message of the DestroyNCO operation previously requested. The confirmation message can be correlated to the correct ADestroyNCOReq message by use of the RequestID.

Parameters:

Name	Provided	Comment
RequestID	C	Provided if supplied on request message.
NCOID	M	Identify the NCO on which the Destroy operation was requested.
CompletionStatus	M	Completion status of the DestroyNCO operation of the NAF.

Related: ADestroyNCOReq.

5.4.6 AGetNCOInfoReq

Class: 1 (Basic Class).

Description: Request message for getting information about an NCO. Each NCO is characterised by some attributes (see Administration Attribute set parameters subclause 5.7.53) which are accessible from the PUF thanks to this request and its confirmation.

Parameters:

Name	Required	Comment
NCOID	M	Identifier of NCO requested on.

Related: AGetNCOInfoCnf.

5.4.7 AGetNCOInfoCnf

Class: 1 (Basic Class).

Description: Confirmation message sent by the NAF to the PUF for answering an AGetNCOInfoReq. It contains the information (see Administration Attribute set parameters subclause 5.7.53) relevant for the requested NCO.

Parameters:

Name	Provided	Comment
NCOID	M	Identifier of NCO requested on.
CompletionStatus	M	Completion status of the GetNCOInfo operation of the NAF.
AAttribute	C	Administration Plane attribute set parameters if CompletionStatus Success else absent.

Related: AGetNCOInfoReq.

5.4.8 AErrorInd

Class: 1 (Basic Class).

Description: This message is related to administrative (i.e. non-protocol related) checking of messages.

Parameters:

Name	Provided	Comment
RequestID	C	Provided if supplied on request message.
CompletionStatus	M	Value indicating the error that has occurred.

Related: None.

5.4.9 ASecurityReq

Class: 2 (Additional class).

Description: This message allows the PUF to engage a security algorithm provided by the NAF. The PUF shall provide the NCOID of the connection it wants to have the security algorithm applied to. The PUF can indicate any connection for security to be applied to. The PUF shall be informed by the NAF with a ASecurityCnf message if it is possible to use security on the indicated connection.
The ASecurityReq message does not state how the connection is secured, or which type of information inside the connection shall be affected by the security algorithm. It is up to the security algorithm to handle the securing of the connection.
The Algorithm parameter indicates to the NAF which security algorithm shall be used to secure the connection. The security algorithm is identified by its name. The names of the available algorithms can be obtained using the Property information provided by the NAF. By using the name "nosecurity" for this parameter, the PUF can indicate that security is no longer needed for the connection.
The optional Key parameter is used by the PUF to give relevant information for the security algorithm to the NAF. The parameter is optional because the security algorithm may or may not need specific information to be activated. The kind of information to be used for the Key parameter is dependent of the security algorithm activated.

Parameters:

Name	Required	Comment
RequestID	O	Request identifier, generated by the PUF.
NCOID	M	Identify the connection for which security has to be activated.
Algorithm	M	The name of the security algorithm to use.
Key	O	Key to use for the security Algorithm.

Related: ASecurityCnf.

5.4.10 ASecurityCnf

Class: 2 (Additional class).

Description: Confirmation message sent to the PUF by the NAF upon completion of the ASecurityReq. The RequestID correlates this confirmation message to the corresponding ASecurityReq.
The CompletionStatus Success indicates that the required security algorithm has been activated or stopped for the requested connection, otherwise the reason for non-activation of the security algorithm is returned. The reason is algorithm specific.

Parameters:

Name	Provided	Comment
RequestID	C	Provided if supplied on request message.
CompletionStatus	M	Completion status of the ASecurity operation of the NAF.

Related: ASecurityReq.

5.4.11 AManufacturerReq

Class: 3 (Additional class).

Description: This message allows the PUF to request the NAF to provide a private manufacturer functionality.
This is the way to handle private functionality not provided by Profile A.

Parameters:

Name	Required	Comment
RequestID	M	Request Identifier.
ManufacturerCode	M	Identifies the manufacturer code (provided by the manufacturer).

Remark: Information about the functionality is mandatory. It is not provided as a parameter of the message but is contained in the data buffer.

Related: None.

5.4.12 AManufacturerInd

Class: 3 (Additional class).

Description: This message gives specific information to a PUF dealing with the requested functionality. The NAF is only allowed to issue manufacturer indications, when the PUF has earlier issued at least one manufacturer private request.

Parameters:

Name	Provided	Comment
RequestID	M	Request Identifier.
ManufacturerCode	M	Identifies the manufacturer code (provided by the manufacturer).
CompletionStatus	O	Identifies the result which is manufacturer specific.

Remark: If information is provided, it shall be done not as a parameter of the message but in the data buffer.

Related: AManufacturerReq.

5.4.13 AChangeNCOReq

Class: 4 (Additional class).

Description: This message is used to change parameter(s) of an existing NCO.
Only NCO of C, C/U type may have the NCOType parameter changed. If the NCO refers to an active connection, the NAF changes the protocol only in stable conditions.

Parameters:

Name	Required	Comment
RequestID	O	Request identifier, generated by the PUF.
NCOID	M	NCO to change.
UDirection	O	Determines how the NCO shall be used, for the User Plane.
UAttributeName	O	Name of static User Plane attribute.
UAttribute Parameters	O	User Plane Attribute parameters. Exclusive with UAttributeName; see the relevant User Plane protocol subclauses for more details.
NCOType	O	Specification of the new NCO type.
UAddress parameters	C	User Plane address; see the Profile A relevant User Plane protocol subclauses for more details.

Related: AChangeNCOCnf.

5.4.14 AChangeNCOCnf

Class: 4 (Additional class).

Description: Confirmation message sent by the NAF to the PUF for answering a AChangeNCOReq. If successful, the changes requested are immediately operational.

Parameters:

Name	Required	Comment
RequestID	O	Request identifier, generated by the PUF.
NCOID	M	NCO to change.
CompletionStatus	M	Completion status of the change NCO operation.

Related: AChangeNCOReq.

5.5 Control Plane messages

5.5.1 Introduction

5.5.1.1 Control Messages classes

The Control Plane messages are divided into eight classes:

1) connection establishment and connection breakdown;
2) overlap sending specific messages;
3) user-to-user information transfer;
4) adjournment of calls;
5) facility invocation;
6) external equipment;
7) addition information;
8) DTMF

As described in subclause 5.2.1.4, not all these classes may be accessible through Profile A. A NAF may choose to implement only some categories from the above list. The error mechanism to indicate to the PUF that a message is not available is described in subclause 5.9.

A PUF can only rely on the availability of the class 1 messages. The availability of other classes of message is NAF dependent.

Table 13 gives an overview of Control Plane messages, the class they belong to and their message identifier.

Table 13: Control Plane messages

Mess. Identif.	Class	Message Name	Purpose of Message
201	1	CAlertReq	State the compatibility with the incoming call.
202	1	CAlertInd	The called terminal states that it may handle a call.
203	1	CConnectReq	Initiate an outgoing call.
204	1	CConnectInd	Present an incoming call.
205	1	CConnectRsp	Accept an incoming call.
206	1	CConnectCnf	Indicate acceptance of an outgoing call by the called terminal.
207	1	CDisconnectReq	Remove a connection or refuse an incoming call.
208	1	CDisconnectInd	Indicate the connection has been removed or the outgoing call has been refused.
209	1	CDisconnectRsp	Confirm the end of a connection.
210	1	CDisconnectCnf	Indicate the other terminal has ended the connection.
212	1	CProgressInd	Indicate a B-channel is connected.
214	1	CStatusInd	Indicate a protocol error.
216	2	CSetupAckInd	Indicate more information is required to proceed the outgoing call.
217	2	CConnectInfoReq	Send more information to process the call.
218	2	CProceedingInd	Indicate no more establishment information shall be accepted for this call.
219	3	CUserInformationReq	Send user-to-user information.
220	3	CUserInformationInd	Present received user-to-user information.
221	3	CCongestionControlReq	Apply flow control operations to user-to-user information exchange.
222	3	CCongestionControlInd	Indicate flow control operation to be applied to user-to-user information exchange.

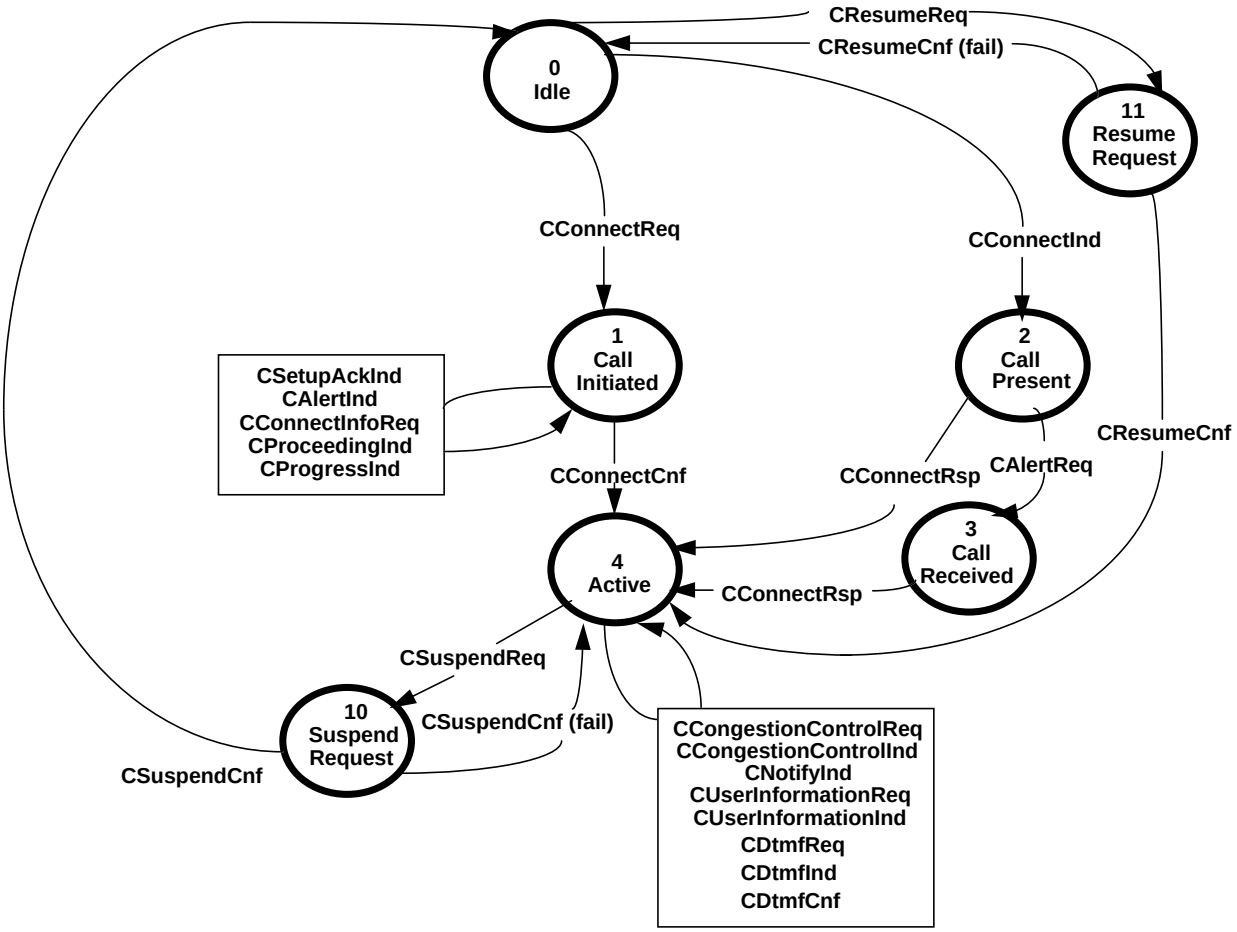
(continued)

Table 13 (concluded): Control Plane messages

Mess. Identif.	Class	Message Name	Purpose of Message
223	4	CSuspendReq	Suspend a connection.
224	4	CSuspendCnf	Response to the demand for suspending a connection.
225	4	CResumeReq	Resume a suspended connection.
226	4	CResumeCnf	Response to the demand for resuming a connection.
228	4	CNotifyInd	Indicate a new state for a connection.
229	5	CFacilityReq	Request a facility from the network.
230	5	CFacilityInd	Indicate a facility coming from the network.
232	6	CExtEquipAvailabilityInd	Indicate that the external equipment is or is not connected to the NAF.
234	6	CExtEquipBlockDiallingInd	Indicate that the call is completely initiated by the external equipment (block dialling).
236	6	CExtEquipKeyPressedInd	Provide to the PUF the code of a depressed key.
238	6	CExtEquipOffHookInd	Indicate that the handset is off-hook.
240	6	CExtEquipOnHookInd	Indicate that the handset is on-hook.
241	7	CAddInfoReq	Request to send additional information related to a call
242	7	CAddInfoInd	Indicate that additional information related to a call has been received.
243	8	CDtmfReq	Initiate, request or stop sending DTMF digits.
244	8	CDtmfCnf	Give the result of a CDtmfReq message.
245	8	CDtmfInd	Indicate that DTMF digits are received.

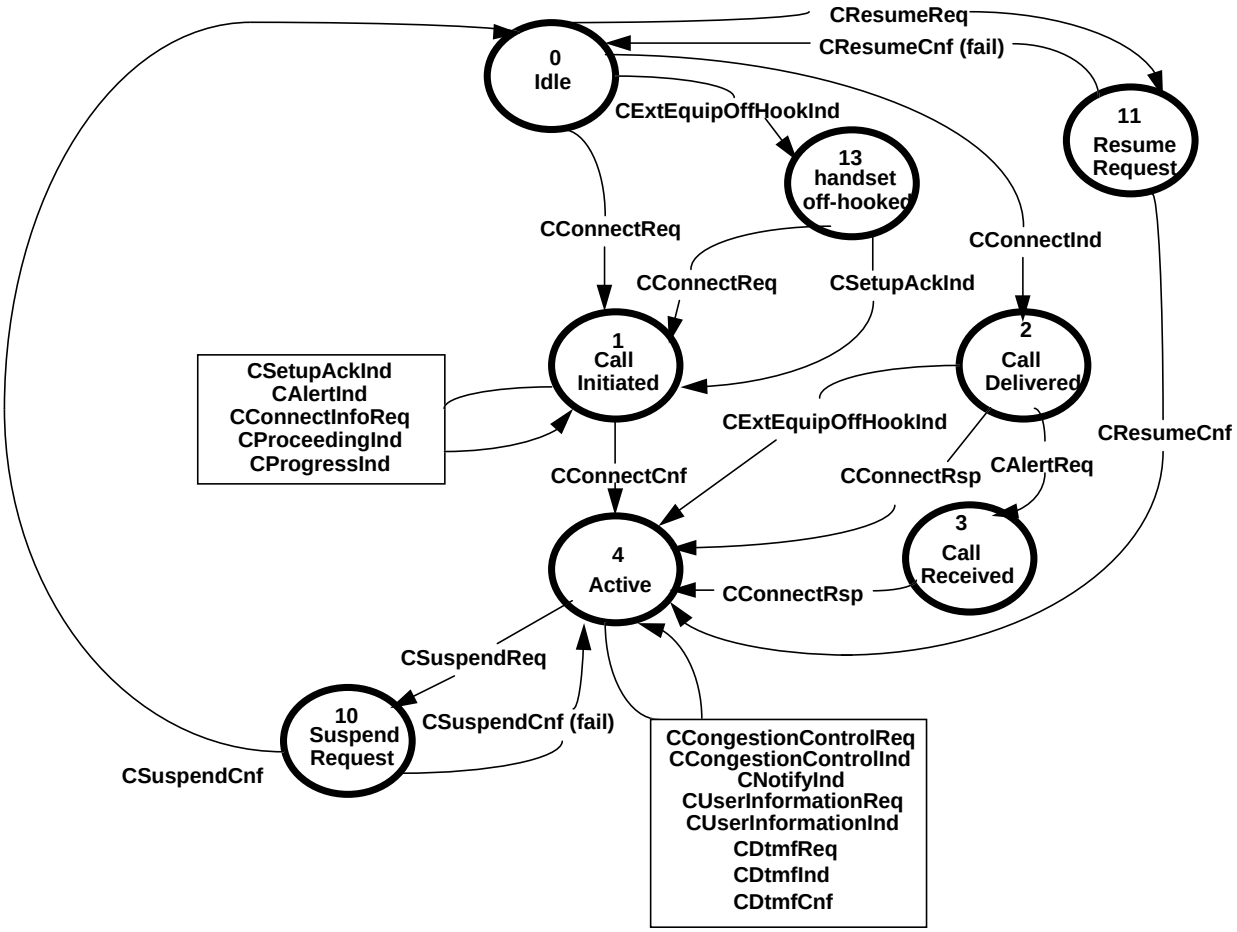
5.5.1.2 Sequencing of Control Plane messages

Figures 7 to 10 present the state diagrams affecting the state of a PUF connection.



NOTE: CExtEquipavailabalityInd can be used in all states. It causes a transition to state 0 if the external equipment is unavailable.

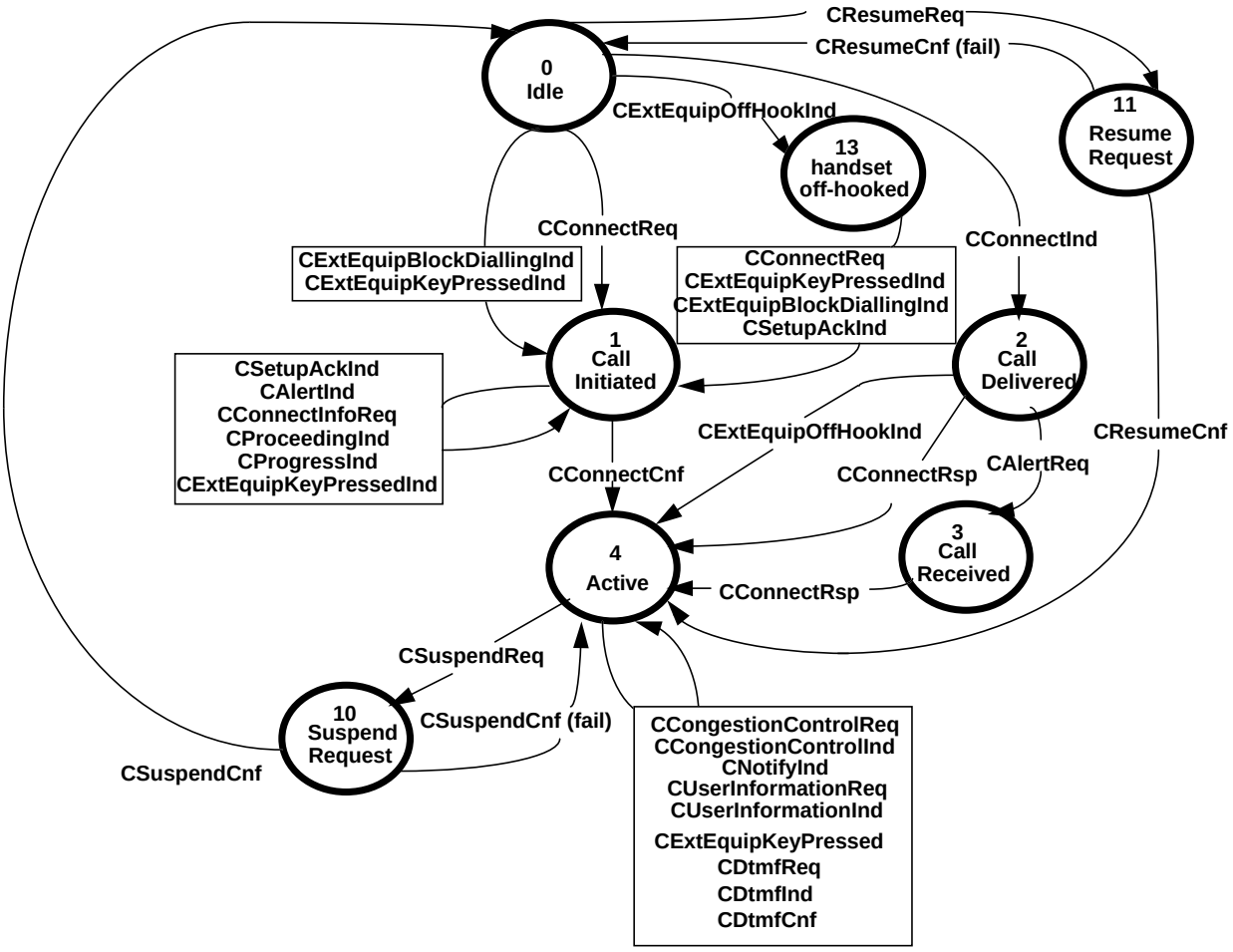
Figure 7: State diagram of a Control Plane no external equipment or external equipment type 1



NOTE 1: CExtEquipOnHookInd can be used in all states except 0. It causes a transition to state 0.

NOTE 2: CExtEquipAvailabalityInd can be used in all states. It causes a transition to state 0 if the external equipment is unavailable.

Figure 8: State diagram of a Control Plane external equipment type 2 or 3

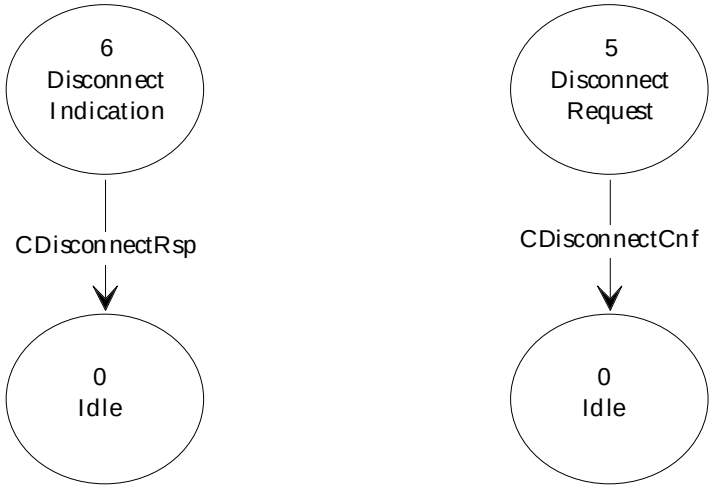


NOTE 1: CExtEquipOnHookInd can be used in all states except 0. It causes a transition to state 0.

NOTE 2: CExtEquipAvailabalityInd can be used in all states. It causes a transition to state 0 if the external equipment is unavailable.

NOTE 3: In figures 7, 8 and 9 if the PUF reaches the Idle state (state 0) by receiving a CSuspendCnf the connection is suspended and may be reused. The NCO however still describes the interaction between PUF and NAF for this connection and cannot be reused. The PUF shall use the NCO again when the connection is resumed or disconnected.

Figure 9: State diagram of a Control Plane external equipment type 4 or 5



NOTE 1: Figures 7 to 10 do not provide any information on the user-to-user information transfer. These messages, depending on the user-to-user service level, do not affect the state of a call from the PUF point of view.

NOTE 2: In order to simplify the interface, a filtering functionality might be added; using this functionality, the PUF may select the subset of messages handled. A detailed description of this functionality is for further study.

Figure 10: State diagram of a Control Plane connection: disconnection

Remarks: CDisconnectInd can be used in all states except 0, 5 and 6. It causes a transition to state 6. CDisconnectReq can be used in all states except 0, 5 and 6. It causes a transition to state 5.
Related network messages and complementary intermediate states can be found in annex B.
Any CStatusInd message may cause a transition to state 0, if provided by the network.
CAddInfoReq or CAddInfoInd message don't change the state of a NCO.

5.5.2 CAlertReq

Class: 1 (Basic class).

Description: This message allows a PUF to indicate its compatibility with an incoming call.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
Facility	O	Supplementary services operation or information.
ProgressIndicator	O	Details concerning call progress.
UserToUserInfo	O	Information to be exchanged between ISDN users.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectReq, CConnectInd, CConnectRsp, CConnectCnf, CAlertInd.

5.5.3 CAlertInd

Class: 1 (Basic Class).

Description: The PUF receives this message when the called terminal has indicated its compatibility.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Identification of the channel used.
Facility	O	Supplementary services operation or information.
ProgressIndicator	O	Details concerning call progress.
Display	O	Information provided by the Network to be displayed.
Signal	O	Information provided by the Network regarding tones.
UserToUserInfo	O	Information to be exchanged between ISDN users.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectReq, CConnectInd, CConnectRsp, CConnectCnf, CAlertInd.

5.5.4 CConnectReq

Class: 1 (Basic Class).

Description: This message is sent by the PUF to initiate an outgoing call. The call shall be initiated to the remote address. This address may be either specified in the message, or have been specified in the address parameters used to create the referenced NCO.
The PUF shall specify the BearerCap parameter to indicate which type of bearer channel is needed. This parameter shall be specified in the message, or have been specified in the attribute parameters used to create the referenced NCO.
The PUF may specify the LLC and HLC parameters, to indicate what type of lower layer and higher layer protocols shall be used for this call.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call. This information is provided by the PUF.
CallingNumber	O	Local address (note 1).
CallingSubaddress	O	Local subaddress (note 1).
CalledNumber	O	Remote address (notes 1 and 2).
CalledSubaddress	O	Remote subaddress (notes 1 and 2).
ChannelIdentification	O	Used by PUF to indicate type of requested Channel. See ChannelIdentification parameter in subclause 5.7.12 for details of supported values. If not provided default is any channel (note 1).
BearerCap	O	Transmission capability required from channel.(note 1).
LLC	O	Lower Layer Compatibility information element (note 1).
HLC	O	High Layer Compatibility information element (note 1).
Keypad	O	Keypad facility information element.
Facility	O	Supplementary services operation or information.
UserToUserInfo	O	Information to be exchanged between ISDN users.
ControllerID	O	Identify a controller. If not provided, the NAF will take the NCO one, if any...
NumberComplete	O	Indicates this message contains the last part of the called number from a PUF point of view (note 3).

NOTE 1: Can be supplied during the creation of NCO. If specified on both message and within the NCO, then parameter specified on message is used and NCO parameter is ignored.
NOTE 2: Either a CalledNumber or a CalledSubaddress parameter - in the message or in during the NCO creation - shall be supplied except in case of overlap sending.
NOTE 3: If this parameter is included, no more CConnectInfoReq messages will be accepted by the NAF for this call.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectCnf, CAlertReq, CAlertInd, CConnectInfoReq.

5.5.5 CConnectInd

Class: 1 (Basic Class).

Description: This message offers an incoming call to all appropriate PUFs (see subclause 6.4.7.1). At this point the call is in the establishment phase, no connection has been established yet.
The number of the calling user may be available to the PUF. If so, it shall be represented in the parameters CallingNumber and CallingSubaddress.
The PUF may receive the parameters BearerCap, LLC, HLC which shall indicate:

- what type of bearer channel shall be used;
- what type of lower layer protocols shall be used for this call;
- what type of higher layer protocols shall be used for this call.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ChannelIdentification	O	Identification of the channel used.
CallingNumber	O	Remote address.
CallingSubaddress	O	Remote subaddress.
CalledNumber	O	Local address.
CalledSubaddress	O	Local subaddress.
BearerCap	O	Network physical resource provided.
LLC	O	Lower Layer Compatibility information element.
HLC	O	High Layer Compatibility information element.
DateTime	O	Date and Time.
Facility	O	Supplementary services operation or information.
Display	O	Information provided by the Network to be displayed.
Signal	O	Information provided by the Network regarding tones
UserToUserInfo	O	Information to be exchanged between ISDN users.
ControllerID	C	Identify the controller on which the call was presented. Absent if only one controller installed. Mandatory otherwise.

Remark: When a PUF receives the No Channel Available information, it can clear or suspend a call to provide a free channel if it wishes to establish a connection.

The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectReq, CConnectRsp, CConnectCnf, CAlertReq, CAlertInd.

5.5.6 CConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept an incoming call. After sending this message, the channel is considered to be established.
The PUF can supply a new value for the LLC, if it is negotiating LLC values.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Used by PUF to indicate type of requested Channel. See ChannelIdentification parameter in subclause 5.7.12 for details of supported values. A value can be provided if the B-channel chosen by the PUF is not the same as those the NAF presents.
LLC	O	Lower Layer Compatibility information element.
Facility	O	Supplementary services operation or information.
ProgressIndicator	O	Details concerning call progress.
UserToUserInfo	O	Information to be exchanged between ISDN users.
ConnectedNumber	O	Part of the remote address.
ConnectedSubaddress	O	Part of the remote address.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectReq, CConnectInd, CConnectCnf, CAlertReq, CAlertInd.

5.5.7 CConnectCnf

Class: 1 (Basic Class).

Description: This message is the response from the called party, indicating it accepts the call. When the PUF receives this message, a channel is considered to be established.
If values for LLC are being negotiated, a new value for the LLC parameter may be supplied in this message.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Identification of the channel used.
LLC	O	Lower Layer Compatibility information element.
DateTime	O	Date and Time.
Facility	O	Supplementary services operation or information.
Display	O	Information provided by the Network to be displayed.
ProgressIndicator	O	Details concerning call progress.
Signal	O	Information provided by the Network regarding tones
UserToUserInfo	O	Information to be exchanged between ISDN users.
ConnectedNumber	O	Part of the remote address.
ConnectedSubaddress	O	Part of the remote address.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CConnectReq, CConnectInd, CConnectRsp, CAlertReq, CAlertInd.

5.5.8 CDisconnectReq

Class: 1 (Basic Class).

Description: This message allows the PUF to initiate the disconnection of a connection or refuse a call.
This message shall be acknowledged by a CDisconnectCnf.
The PUF may indicate the reason to disconnect a connection or refuse a call by supplying the CauseToNAF parameter.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
CauseToNAF	O	PUF reason to disconnect the call. If not provided by the PUF, the #16 "Normal Call Clearing" cause shall be provided by the NAF.
Facility	O	Supplementary services operation or information.
UserToUserInfo	O	Information to be exchanged between ISDN users.

Remarks: The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CDisconnectInd, CDisconnectRsp, CDisconnectCnf.

5.5.9 CDisconnectInd

Class: 1 (Basic Class).

Description: This message informs the PUF that the remote user has initiated the disconnection of the connection or has refused the call.
The PUF shall acknowledge this message with a CDisconnectRsp.
The CauseToPUF parameter shall indicate the reason for disconnecting or refusing.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CauseToPUF	M	Reason why the call is being disconnected. If not provided by the Network the NAF shall introduce the #16 "Normal Call Clearing" cause. See also the remark.
Facility	O	Supplementary services operation or information.
Display	O	Information provided by the Network to be displayed.
ProgressIndicator	O	Details concerning call progress.
Signal	O	Information provided by the Network regarding tones
UserToUserInfo	O	Information to be exchanged between ISDN users.

Remarks: The network shall only transfer one cause to the NAF, so the PUF shall only receive one cause.
The availability of UserToUserInfo depends on the user-to-user service level. See subclause 5.5.38 for details on user-to-user information.

Related: CDisconnectReq, CDisconnectRsp, CDisconnectCnf.

5.5.10 CDisconnectRsp

Class: 1 (Basic Class).

Description: With this message, the PUF acknowledges that a connection has ended or a call has been refused. From the point of view of the PUF the channel is now cleared, and the NCOID may be reused by the NAF.
This message is sent by the PUF to acknowledge the CDisconnectInd.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
Facility	O	Supplementary services operation or information.

Related: CDisconnectReq, CDisconnectInd, CDisconnectCnf.

5.5.11 CDisconnectCnf

Class: 1 (Basic Class).

Description: With this message, the PUF is informed that the connection has ended or a call has been refused, and the channel has been cleared down. The NCOID may now be reused by the NAF.
This message is the acknowledgement by the remote user or by the network of the CDisconnectReq.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CauseToPUF	O	Reason why a supplementary service request has been rejected by the network.
Facility	O	Supplementary services operation or information.
Display	O	Information provided by the network to be displayed.
Signal	O	Information provided by the network regarding tones

Related: CDisconnectReq, CDisconnectInd, CDisconnectRsp.

5.5.12 CProgressInd

Class: 1 (Basic Class).

Description: The PUF receives this message when information is available in the B-channel or in case of internetworking. The channel shall be connected.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Identification of the channel used.
CauseToPUF	O	Reason for the message.
Display	O	Information provided by the network to be displayed.
ProgressIndicator	M	Details concerning call progress.
UserToUserInfo	O	Information to be exchanged between ISDN users.

Related: CConnectReq, CConnectInd, CConnectRsp, CConnectCnf, CAlertInd.

5.5.13 CStatusInd

Class: 1 (Basic Class).

Description: With this message, the PUF shall be informed that a signalling protocol error, as defined in subclause 6.4.8.8, has occurred.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CauseToPUF	M	Identifies the protocol error that has occurred.

Related: None.

5.5.14 CSetupAckInd

Class: 2 (Additional Class).

Description: The PUF receives this message when more establishment information is needed to perform the call, in the overlap sending case.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Identification of the channel used.
Display	O	Information provided by the network to be displayed.
ProgressIndicator	O	Details concerning call progress.

Related: CConnectInfoReq, CConnectReq.

5.5.15 CConnectInfoReq

Class: 2 (Additional Class).

Description: This message allows a PUF to use the overlap sending technique for connection establishment. Overlap sending means that the PUF supplies the address information in more than one step: a CConnectReq message with incomplete address information may be followed by several CConnectInfoReq messages until the address is complete. This mechanism is similar to dialling on a keypad.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
CalledNumber	M	Part of the remote address (note 1).
NumberComplete	O	Indicates this message contains the last part of the called number from a PUF point of view (note 2).
NOTE 1:	With each CConnectInfoReq message the NAF accumulates the address information. The PUF does not indicate that the address information is complete; this can implicitly be concluded from the receipt of a CProceedingInd message. A Subaddress can only be specified in the CConnectReq message. This is due to the restrictions imposed by the D-channel protocol (SETUP network message).	
NOTE 2:	If this parameter is included, no more CConnectInfoReq messages shall be accepted by the NAF for this call.	

Remarks: The PUF shall have sent a CConnectReq message with the first part of the called information prior to this message.

Related: CConnectReq, CProceedingInd, CSetupAckInd.

5.5.16 CProceedingInd

Class: 2 (Additional Class).

Description: The PUF receives this message when no more establishment information is accepted, in the overlap sending case. As the network may not provide this message, the PUF cannot rely on its reception.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
ChannelIdentification	O	Identification of the channel used.
Display	O	Information provided by the network to be displayed.
ProgressIndicator	O	Details concerning call progress.

Related: CConnectReq, CConnectInfoReq, CSetupAckInd.

5.5.17 CUserInformationReq

Class: 3 (Additional Class).

Description: This message allows a PUF to request user-to-user information be sent on an established connection.
The call state that allows the PUF to send user-to-user information is dependent on the user-to-user service level provided by the network or the subscription. See subclause 5.5.38 for details on user-to-user information.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
MoreData	O	Indicates to the peer entity that another user-to-user message follows.
UserToUserInfo	M	Information to be exchanged between ISDN users.

Remarks: This message is available only if a user-to-user service level 2 and above has been subscribed to. See subclause 5.5.38 for details on user-to-user information.

Related: CUserInformationInd, CCongestionControlReq, CCongestionControlInd.

5.5.18 CUserInformationInd

Class: 3 (Additional Class).

Description: This message allows a NAF to present to the PUF user-to-user information received on an established connection.
The call state that allows the reception of user-to-user information is dependent on the user-to-user service level provided by the network or the subscription. See subclause 5.5.38 for details on user-to-user information.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
MoreData	O	If present, the peer entity indicates that another User to User message follows.
UserToUserInfo	M	Information exchanged between ISDN users.

Remarks: This message is available only if a user-to-user service level 2 and above has been subscribed to. See subclause 5.5.38 for details on user-to-user information.

Related: CUserInformationReq, CCongestionControlReq, CCongestionControlInd.

5.5.19 CCongestionControlReq

Class: 3 (Additional Class).

Description: This message allows a PUF to apply flow control operations on the user-to-user information provided via the CUserInformationInd message.
The flow control operation is only defined to operate on the local side of the connection. The flow control operates using the ready/not ready mechanism. The initial condition for user-to-user information exchange shall be ready. To set the condition for flow control the PUF shall set the parameter CongestionLevel to the appropriate value.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
CongestionLevel	M	Flow control value.
CauseToNAF	O	Include if information lost.
NOTE: This message in available only if a user-to-user service level 2 and above has been subscribed to. See subclause 5.5.38 for details on user-to-user information.		

Remarks: For the flow control provided by this message, ready is assumed as the initial status. The flow control for each direction shall be operated independently. This message has only a local meaning.

Related: CUserInformationReq, CUserInformationInd, CCongestionControlInd.

5.5.20 CCongestionControlInd

Class: 3 (Additional Class).

Description: This message allows a NAF to indicate to a PUF that a flow control operations has been applied to the user-to-user information provided via the CUserInformationReq message.
The flow control operation is only defined to operate on the local side of the connection. The flow control operates using the ready/not ready mechanism. The initial condition for user-to-user information exchange shall be ready. The parameter CongestionLevel shall give the new value for the flow control on the user-to-user information exchange to the PUF.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CongestionLevel	M	Flow control value.
CauseToPUF	O (note)	Include if information is lost.
Display	O	Information provided by the network to be displayed.
NOTE: The network shall only transfer one cause to the NAF, so the PUF shall only receive one cause.		

Remarks: This message is available only if a user-to-user service level 2 and above has been subscribed to.
For the flow control provided by this message, ready is assumed as being the initial status. This message has only a local meaning. The flow control for each direction shall be operated independently.

Related: CUserInformationReq, CUserInformationInd, CCongestionControlReq.

5.5.21 CSuspendReq

Class: 4 (Additional Class).

Description: This message allows a PUF to suspend, but not to disconnect, a connection.
After sending this message, the PUF shall be informed if the connection is suspended.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.

Remarks: Using this message in conjunction with running a protocol on the connection is the responsibility of the PUF.
When suspending a connection, it is not guaranteed that the connection can subsequently be resumed.

Related: CSuspendCnf, CResumeReq, CResumeCnf, CNotifyInd.

5.5.22 CSuspendCnf

Class: 4 (Additional Class).

Description: This message is the answer to a CSuspendReq message. The NAF provides the PUF with the result of its suspend request.
The parameter Response shall indicate if the connection is suspended.
If the PUF receives a CSuspendCnf the connection is suspended and may be reused. The NCO, however, still describes the interaction between PUF and NAF for this connection and cannot be reused. The PUF shall have to use the NCO again when the connection is resumed or disconnected.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CompletionStatus	M	Indicates the state of the suspension: - success: if the suspension is accepted; - operation failed: if the suspension is refused.
CauseToPUF	C (note)	Mandatory if case of suspension refused, it indicates the reason why the suspension was refused. Absent in case of success.
Display	O	Information provided by the network to be displayed.
NOTE: The network shall only transfer one cause to the NAF, so the PUF shall only receive one cause.		

Related: CSuspendReq, CResumeReq, CResumeCnf, CNotifyInd.

5.5.23 CResumeReq

Class: 4 (Additional Class).

Description: This message allows a PUF to resume, i.e. a suspended connection is reconnected.
After sending this message, the PUF shall be informed if the suspended connection is reconnected.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.

Related: CSuspendReq, CSuspendCnf, CResumeCnf, CNotifyInd.

5.5.24 CResumeCnf

Class: 4 (Additional Class).

Description: This message is the answer to an CResumeReq message. The NAF provides the PUF with the result of its resume request.
The response parameter shall indicate if the connection is resumed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
CompletionStatus	M	Indicates the state of the resume operation: - Success: if the operation succeed; - OperationFailed: if the operation failed.
CauseToPUF	C (note)	Mandatory if case of operation failure, it indicates the reason why the operation was refused. Absent in case of success.
Display	O	Information provided by the network to be displayed.
NOTE: The network shall only transfer one cause to the NAF, so the PUF shall only receive one cause.		

Remarks: The result for resuming a connection might be negative (OperationFailed) if the NAF or the network does not have resources available, i.e. channels, to reconnect the connection.

Related: CSuspendReq, CSuspendCnf, CResumeReq, CNotifyInd.

5.5.25 CNotifyInd

Class: 4 (Additional Class).

Description: This message is provided by the NAF to indicate to the PUF a new state for the connection.
As example, this message may be issued if the remote user suspends or resumes a connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call.
NotificationIndicator	M	New state.
Display	O	Information provided by the Network to be displayed.

Related: CSuspendReq, CSuspendCnf, CResumeReq, CResumeCnf.

5.5.26 CFacilityReq

Class: 5 (Additional Class).

Description: This message allows the PUF to request a facility from the network. This facility may or may not be related to an established connection.
For details on the use of facility messages and parameters and the coding of the facility parameter refer to subclause 5.7.32.

Parameters:

Name	Required	Comment
NCOID	O	Provided by the PUF if the facility is related to an established connection.
Facility	M (note)	Supplementary services operation or information.
NOTE: If the PUF supplies transparent coding of the facility information element, all information following this transparent coding shall be handed back transparently.		

Related: CFacilityInd.

5.5.27 CFacilityInd

Class: 5 (Additional Class).

Description: This message provides to the PUF the facility coming from to the network. This facility may or may not be related to an established connection.
For details on the use of facility messages and parameters and the coding of the facility parameter refer to subclause 5.7.32.

Parameters:

Name	Provided	Comment
NCOID	O	Provided by the NAF if the facility is related to an established connection.
Facility	M (note)	Supplementary services operation or information.
Display	O	Information provided by the network to be displayed.
NOTE: If the PUF has supplied transparent coding of the facility information element, all information following this transparent coding shall be handed back transparently.		

Related: CFacilityReq.

5.5.28 CExtEquipAvailabalityInd

Class: 6 (Additional Class).

Description: With this message, the PUF is informed about the availability of the external equipment. When a connection is active, if the external equipment becomes unavailable the NAF is in charge of breaking down the communication.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ExtEquipAvailability	M	Indicates the external equipment availability.
ControllerID	C	Identify the controller. Absent if only one controller installed else mandatory.

Related: None.

5.5.29 CExtEquipBlockDiallingInd

Class: 6 (Additional Class).

Description: With this message, the PUF gets the dialling information input by the user with the keypad of the external equipment in the case of a block sending. This message contains the complete remote address and/or the remote subaddress.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ExtEquipBlockDialling	M	Provides to the PUF the remote address and/or subaddress in the case where the external equipment allows the block sending.
ControllerID	C	Identify the controller. Absent if only one controller installed else mandatory.

Related: None.

5.5.30 CExtEquipKeyPressedInd

Class: 6 (Additional Class).

Description: With this message, the PUF gets the dialling information input by the user with the keypad of the external equipment in the case of an overlap sending. One message is provided to the PUF for each key pressed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ExtEquipKeyPressed	M	Provides to the PUF the code of the pressed key if the external equipment dials in the overlap sending mode.
ControllerID	C	Identify the controller. Absent if only one controller installed else mandatory.

Related: None.

5.5.31 CExtEquipOffHookInd

Class: 6 (Additional Class).

Description: With this message, the PUF is informed that the handset of the external equipment is off-hook. Depending on the type of external equipment and on the current state of the connection, this message can be interpreted in different ways (see figures 7, 8 and 9).

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ControllerID	C	Identify the controller. Absent if only one controller installed else mandatory.

Related: CExtEquipOnHookInd.

5.5.32 CExtEquipOnHookInd

Class: 6 (Additional Class).

Description: With this message, the PUF is informed that the handset of the external equipment is on-hook. Depending on the type of external equipment and on the current state of the connection, this message can be interpreted according to different ways (see figures 7, 8 and 9).

Name	Provided	Comment
NCOID	M	Identifies the call. This information element is provided by the NAF.
ControllerID	C	Identify the controller. Absent if only one controller installed else mandatory.

Related: CExtEquipOffHookInd.

5.5.33 CAddInfoReq

Class: 7 (Additional Class).

Description: This message allows a PUF to send additional information related to a call. The overlap sending information is handled via the class 2 CConnectInfoReq message. This message may be used to convey network related information (e.g. particular identification procedure).

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
AdditionalInformation	M	Conveys network related information.

Related: CAddInfoInd.

5.5.34 CAddInfoInd

Class: 7 (Additional Class).

Description: This message allows a NAF to send additional information related to a call provided by the network (e.g. particular identification procedure).

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
Display	O	Information provided by the network to be displayed.
AdditionalInformation	M	Conveys network related information.

Related: CAddInfoReq.

5.5.35 CDtmfReq

Class: 8 (Additional Class).

Description: This message is intended to provide to the PUF the ability to start or stop DTMF listen on B-channel data or send DTMF digits. The B-channel shall be connected

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
RequestID	O	Identifies an ordered operation.
DtmfOperation	M	Indicates to start or stop DTMF listening or DTMF digits sending.
DtmfToneDuration	O	Time in ms for one digit, default is 40 ms.
DtmfGapDuration	O	Time in ms between the digits, default is 40 ms.
DtmfDigits	O	Characters to send.
NOTE: The PUF should start DTMF listening to receive DTMF digits.		

Related: CDtmfInd.

5.5.36 CDtmfCnf

Class: 8 (Additional Class).

Description: This message is intended to provide to the NAF the ability to acknowledge or not a CDtmfReq message.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
RequestID	C	Mandatory if present in the previous operation, else absent.
DtmfResult	M	Acknowledge of CDtmfReq message.
NOTE: Due to the asynchronism of the message exchanges between the PUF and the NAF, a PUF may identify a previous CDtmfReq message via the RequestID parameter if included.		

Related: CDtmfReq.

5.5.37 CDtmfInd

Class: 8 (Additional Class).

Description: This message is intended to provide to the PUF the ability to receive DTMF digits on B-channel data. The channel shall be connected.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the call.
DtmfDigits	M	Received characters.
NOTE: The PUF should start a DTMF listening operation via a CDtmfReq message before to receive DTMF digits.		

Related: CDtmfReq.

5.5.38 User to User information exchange

The use of user-to-user information exchange is dependent on the user-to-user service level provided by the network or the subscription.

In ETS 300 102 [2] three user-to-user service levels are defined:

- service 1:
  user-to-user information exchanged during the set-up and clearing phase of a call;
- service 2:
  user-to-user information exchanged during call establishment;
- service 3:
  user-to-user information exchanged while a call is in the active state.

For the PUF, the use of UserToUserInfo parameter inside messages and the UserInformation messages is dependent on the service level.

The following usage of UserToUserInfo parameter and UserInformation messages is defined, relating to the service level:

- Service 1:
  using the UserToUserInfo parameter in:
  CAlertReq;
  CAlertInd;
  CConnectReq;
  CConnectInd;
  CConnectRsp;
  CConnectCnf;
  CDisconnectReq;
  CDisconnectInd;
- Service 2:
  using the CUserInformation messages between the sending/receiving of CAlertReq/Ind and CConnectRsp/Cnf messages.
- Service 3:
  using CUserInformation messages in the active state of a call.

All three services may be used separately or in any combination with a single call.

NOTE: Services 2 and 3 are currently provided using the method described in ETS 300 102 [2].

5.5.39 Implementation of supplementary services

5.5.39.1 Advice of Charge during call (AOC-D)

Description: This supplementary service allows the PUF to obtain charging information during a connection.

Operation: Either at the establishment of the connection the PUF can activate the AOC-D supplementary service, or this service is available for every connection.

During the connection the PUF shall get the subtotal for this connection, either as currency or as charging units. At the end of the connection, the PUF shall obtain the total charging information for the connection.

Implementation: The PUF shall activate the AOC-D supplementary service, by including the corresponding facility field in the connect message.

After activation, the charging information shall be presented by the PUF using the facility field. For the coding of this field see subclause 5.7.32.

Errors shall be reported by using the facility field. The PUF shall not take any protocol action upon receiving an error.

Relevant fields: Facility (subclause 5.7.32).

5.5.39.2 Advice of Charge at End of call (AOC-E)

Description: This supplementary service allows the PUF to obtain charging information at the end of a connection.

Operation: Either at the establishment of the connection the PUF can activate the AOC-E supplementary service, or this service is available for every connection.

At the end of the connection, the PUF shall obtain the total charging information for the connection either as currency or as charging units.

Implementation: The PUF shall activate the AOC-E supplementary service, by including the corresponding facility field in the connect message.

After activation, the charging information shall be presented by the PUF using the facility field. For the coding of this field see subclause 5.7.32.

Errors shall be reported by using the facility field. The PUF shall not take any protocol action upon receiving an error.

Relevant fields: Facility (subclause 5.7.32).

5.6 User Plane

5.6.1 User Plane Protocols Management Architecture

5.6.1.1 Introduction

This clause describes the protocol management provided by the User Plane of Profile A. The User Plane provides operations which facilitate establishment, data exchange and/or release of logical communication channels. It provides messages that allow the use of underlying protocols. It is related to the user connection, which may either be associated with a connection on the B-channel or a data connection on the D-channel.

Profile A is located between layers 3 and 4 of the OSI reference model. In the case of transparent access, the NAF considers layers 2 and 3 as Null layers. In case of link access, the NAF considers layer 3 as Null layer and for network access layers 2 and 3 are implemented as shown in figure 11.

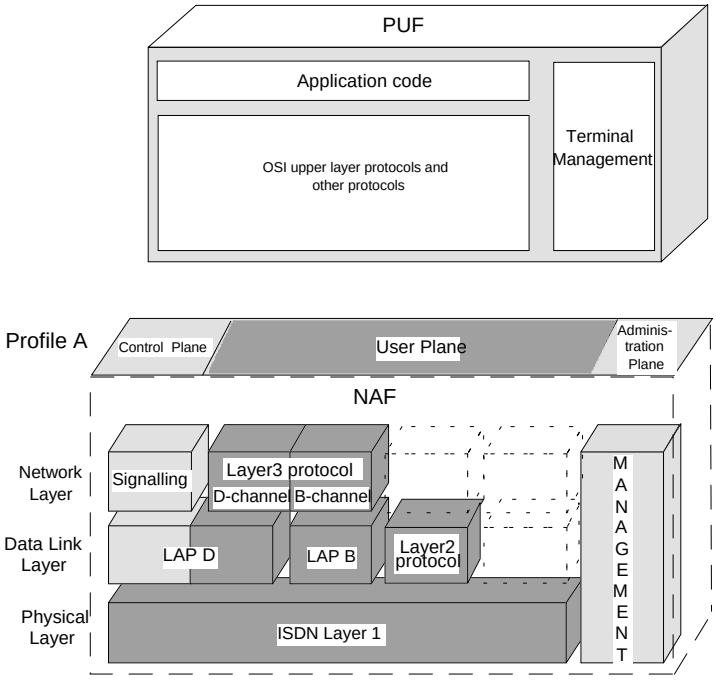


Figure 11: OSI location

For the support of transparent access to the ISDN B-channel, the User Plane provides access to the Physical Service Access Point (Ph-SAP). The User Plane also allows for access of another Service Access Point (SAP). It provides the services defined in ITU-T Recommendation X.213 [6] and is, therefore, located at the Network layer Service Access Point (N-SAP).

Profile A in this ETS specifies the usage of possible protocols for the data transfer service. The User Plane provides the services for a variety of protocols. They all are optional and may be divided into the following groups:

- layer 1 protocols;
- layer 2 protocols;
- layer 3 protocols.

5.6.1.2 Message access

The data management covers the functionality used to:

- establish data connections on already established physical connections, if needed;
- exchange data.

The User Plane of Profile A provides the functionality defined by the data management.

So far, three sets of messages are defined in the User Plane. One set allows access to User Plane protocols providing the OSI network-layer service interface. The second one provides access to link-layer service interface. The last set provides a transparent interface where the PUF implements the protocol to be run over the connection.

For both types of access it is important that there exists a signalling connection before any data access can be granted. In general, establishment of that signalling connection is achieved by use of Control Plane functionality, described in subclause 5.5, whereas the establishment of the data access is achieved, if necessary, using User Plane functionality.

In the following subclauses, the different methods for message access which are supported by Profile A are explained.

5.6.1.2.1 The physical layer access (transparent access)

Profile A supports a transparent message access. It provides access to the transparent layers 2 and 3 (NULL Layers) and thus provides direct access to the physical layer of ISDN, providing a byte synchronised control over a B-channel. The bearer services provided by the network (Bearer Capability) is not limited to digital service. For example, Bearer Capability may be "speech".

As with any message access, this type of message access offers its own set of operations. Table 14 provides an overview on User Plane operations for this message access.

Due to the nature of the access, only operations allowing direct byte stream access are provided for this message access; no user protocol is running. Thus, by establishment of a signalling connection the transparent data access becomes available. Unlike the access via the network layer, only one data connection is accessible per signalling connection.

Table 14: User Plane operations for transparent access

Operation name	**Purpose of operation**
Data	Data transfer
Error	Indicates an error has occurred

When using a connection on the transparent access with an NCO (NCOType C) associated with an external equipment, the data generated on the connection shall be sent to the external equipment rather than used to generate transparent access messages. This case is outside the scope of this ETS.

5.6.1.2.2 The link layer access

Profile A supports a link layer message access. It provides access to the transparent layer 3 (NULL Layer) and thus provides direct access to the link layer of ISDN.

This message access offers its own set of operations. Depending on the user protocol, these operations are available or not. Table 15 provides with an overview of User Plane operations for this message access.

Table 15: User Plane operations for link layer access

Operation name	Purpose of operation
Connect	Establish a peer-to-peer user connection
Data	Exchange data over an established user connection, hereby relying on flow control provided by underlying protocol
Disconnect	Disconnect connection
ReadyToReceive	Control the normal data flow
Error	Indicates an error has occurred

5.6.1.2.3 The network layer access

Profile A supports a network layer message access. It provides access to the User Plane protocols running in the ISDN network layer. Thus, it provides access to a network layer connection.

This message access offers its own set of operations. Depending on the user protocol, these operations are available or not. Table 16 provides an overview of User Plane operations for this message access.

The operational set is based on ITU-T Recommendation X.213 [6].

Profile A provides, for some protocols, co-ordination functionality which removes the need for the PUF to use Control Plane functionality. This co-ordination functionality, which is available to the PUF on demand, implicitly builds a signalling connection when a user connection is requested.

Table 16: User Plane operations for network layer access

Operation name	Purpose of operation
Connect	Establish a peer-to-peer user connection
Data	Exchange data over an established user connection, hereby relying on flow control provided by underlying protocol
Expedited data	Exchange data over an established user connection without relying on flow control provided by underlying protocol
Data acknowledge	Acknowledgement of data reception over an established user connection
Reset	Clearing of data transfer
Disconnect	Disconnect connection
ReadyToReceive (note)	Control the normal data flow
NOTE: This operation is not based on ITU-T Recommendation X.213 [6].	

5.6.1.3 Protocols

5.6.1.3.1 Supported User Plane protocols

There are different user layer protocols which can be accessed. One of these User Plane protocols is selectable at the creation of the NCO.

Table 17 lists whether it is mandatory (M) or optional (O) for each protocol to be supported by a NAF.

Table 17: Supported User Plane protocols

Protocol	NAF supported	Layer
Network layer according to ETS 300 080 [1]	M	3
ISO/IEC 8208 [3]	M	3
Network layer of Recommendation T.70 NL	O	3
T.30	O	3
Null layer 3 with access to ISO 7776 on layer 2	O	2
Null layer 3 with transparent access to HDLC framing	O	2
Null layer 3 with transparent access to HDLC framing with error indication	O	2
PPP	O	2
SDLC	O	2
V.110 asynchronous (note)	O	2
V.110 synchronous (note)	O	2
Transparent B-Channel access with byte framing from the network	M	1
NOTE: A V.110 access is offered to a PUF at level 2 but other ways to use this protocol are possible.		

5.6.1.3.2 Protocol selection

The protocol selection is made during the creation of the NCO, by the use of two parameters: NCOType and UProtocol parameters (see description of the ACreateNCOReq function in subclause 5.4.1).

5.6.1.3.2.1 NCOType parameter

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C (1) - signalling access only (note). U3 (2) - network user access with NAF signalling co-ordination (NAF co-ordination functionality). C/U (3) - signalling and network, link or physical user access. U3G (4) - network user access to additional virtual circuits. This NCO shall be grouped to an already created U3 or C/U type NCO.
NOTE:	NCO type C is outside the scope of the user protocol management.			

5.6.1.3.2.2 UProtocol parameter

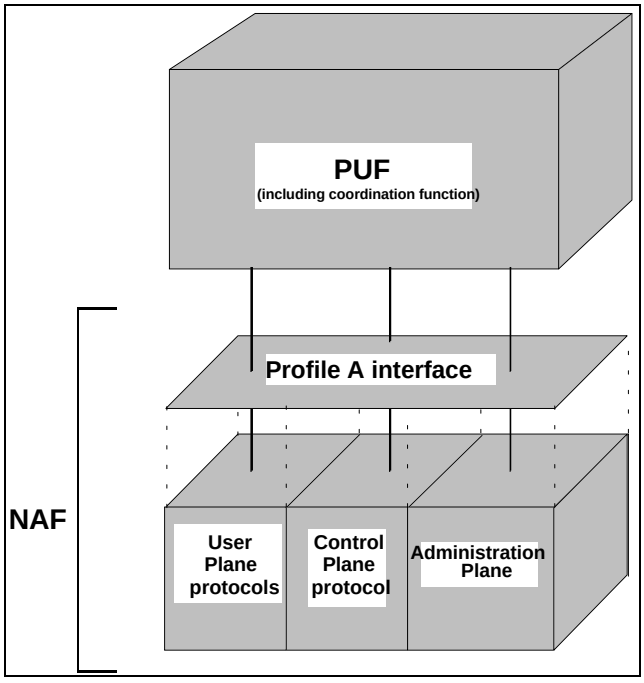
Description: This is used to select the User Plane protocol. If the length is 3, the first octet contains the layer 3 protocol requested, the second octet contains the layer 2 protocol requested and the third octet contains the layer 1 protocol requested.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	Default (255) - T.90 T.90 (1) ISO 8208 (2) T.70 NL (3) NULL (4) T.30 (5)
L2Protocol	Octet	P	C (note 1)	Default (255) - ISO 7776 ISO 7776 (0) Frame oriented transparent (2) Frame oriented transparent with error indication (3) PPP (4) SDLC (5) V.110 asynchronous (6) V.110 synchronous (7) NULL (8)
L1Protocol	Octet	P	C (note 2)	Default (255) - Transparent access with byte framing from the network. Transparent access with byte framing from the network. (1)
NOTE 1: Mandatory if L3Protocol is NULL. NOTE 2: Mandatory if L3Protocol and L2Protocol are NULL.				

5.6.1.4 Co-ordination function

Profile A provides direct access to signalling and to the user connection, associated with the D- and B-channels of the ISDN. A PUF which uses this method, shall handle the establishment of a user connection by using the basic call control provided by the Control Plane. The co-ordination between signalling and user connection is handled only by the PUF. Figure 12 shows the PUF provided co-ordination function. As a result of controlling the signalling connection, the PUF can use the supplementary services.

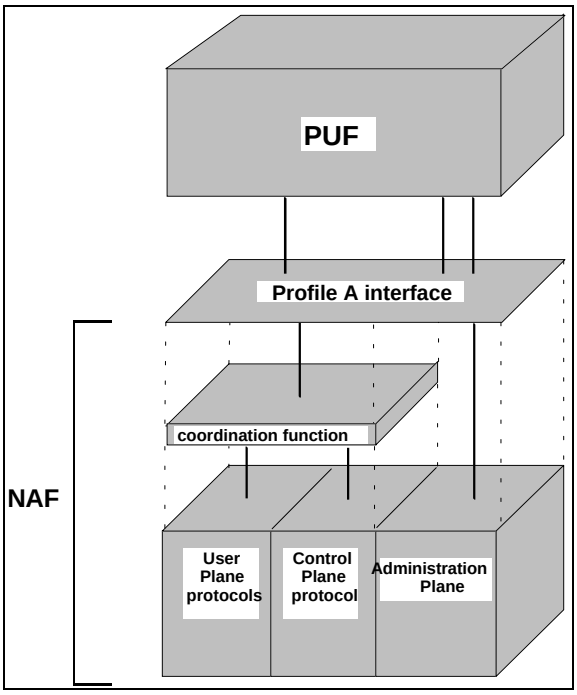


NOTE: The existence of a co-ordination function inside the PUF is outside the scope of this ETS.

Figure 12: PUF co-ordination

The PUF may be offered an ISO Connection-mode Network Service (CONS) as defined in ITU-T Recommendation X.213 [6]. This abstraction is provided by a co-ordination function, which maps the primitives of CONS in the User Plane according to the primitives of the Control Plane and User Plane protocols. The co-ordination function can only be used with the User Plane protocols relating to ITU-T Recommendation X.213 [6]. The co-ordination function is provided as part of the NAF. Since the NAF

manages the co-ordination between signalling and user connection, the PUF shall not access the Control Plane. Figure 13 shows the NAF provided co-ordination function.



NOTE: The co-ordination function is only defined for User Plane protocols related to ITU-T Recommendation X.213.

Figure 13: NAF co-ordination

The co-ordination function does not affect the Administration Plane.

Even if the co-ordination function is used by the PUF, the layer 2 and layer 3 protocols used are the selected protocols for the Network and Link access.

To achieve a connection which is to be NAF co-ordinated, the PUF exchanges the following message:

ACreateNCOReq, with NCOType U3 and the relevant information.

A connection can then be requested using the UConnectReq. All other messages in the User Plane can still be used by the PUF. No Control Plane messages can be used in combination with an NCO of type U3. For the State diagrams, see the other relevant subclauses in 5.6.

The co-ordination function may not be available for all the User Plane protocols. Therefore the co-ordination function availability is noted in each relevant user protocol part.

5.6.1.5 Selection criteria

5.6.1.5.1 NCO Selection: User Plane information element

In order to apply the right NCO on an incoming call, the NAF uses various criteria. General mechanism and Control Plane information elements are described in subclause 5.8.

Some User Plane information elements can also applied for the NCO selection. Useful elements are particular to the protocol in use. For example, in case of ISO 8208 [3], User Plane information elements are the following:

- packet size negotiation;
- window size negotiation.

See subclauses 5.6.2 to 5.6.6 for User Plane information elements to be used.

5.6.1.5.2 Action if no NCO available: User Plane incoming call

A disconnect containing the protocol specific reason is issued by the NAF. If applicable, the exact reason is provided in the protocol relevant subclause.

5.6.1.6 User Plane error checking

Administrative message error information is returned in the Administration Plane error message. For details on the message error handling refer to subclause 5.9.2.

The protocol error detection takes place after administrative checking and the mechanism used to return error information depends of the protocol. These mechanisms are described in relevant subclause of each protocol description.

Invalid length of user data is considered as protocol error.

5.6.1.7 User Plane attribute sets

Attribute sets are used to keep together important parameters for driving user protocols. A collection of attribute sets exists for this plane. They are defined in each protocol relevant subclause.

5.6.2 Layer 1 Protocols

5.6.2.1 Transparent B-channel access with byte framing from the network

5.6.2.1.1 Introduction

This subclause describes the specific elements (messages, parameters, attribute sets,...) relating to the layer 1 user protocols. It covers the Transparent B-channel access protocol with byte framing from the network. The BearerCap parameter indicates if the B-channel is used at 64 kbit/s or 56 bit/s. Other layer 1 user protocols are for further study.

For this access, the NAF considers layers 2 and 3 as a Null layers, as shown in figure 14.

The OSI location of the 64 kbit/s transparent protocol is shown in figure 14.

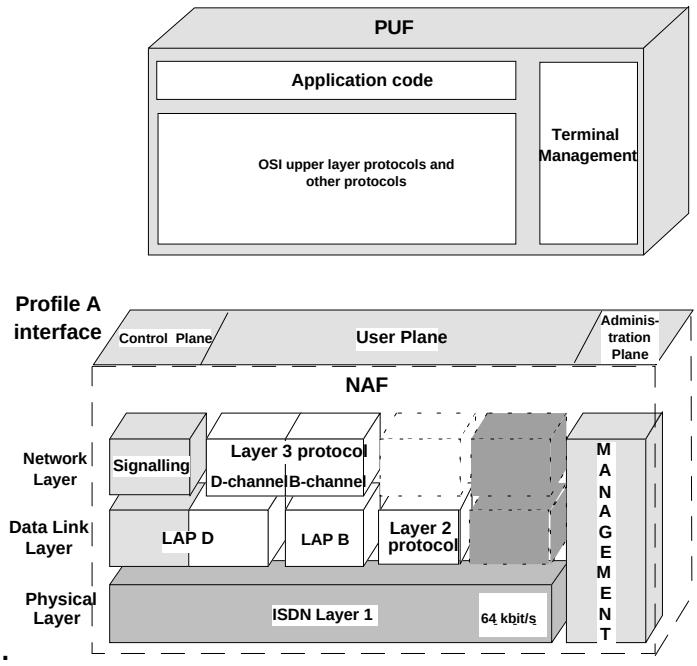


Figure 14: OSI location

General description of conventions is provided in subclause 5.2.3.

5.6.2.1.2 Messages

Table 18 gives an overview of user messages.

Table 18: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
307	1	UDataReq	Request transfer of data.
308	1	UDataInd	Indicate arrival of transferred data.
319	1	UErrorInd	Indicate an error.

5.6.2.1.2.1 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send *transparent* data on the B-channel. By default, data are sent without any protocol as byte stream. The synchronisation used on the B-channel is character oriented. When no more data is available, the NAF will send the IdleFlag octet provided in the Attribute Set used for this connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the Control Plane connection.

Remark: Data to send are mandatory. They are not provided as a parameter of the message. **Mandatory data shall be provided in the data buffer.**

Related: None.

5.6.2.1.2.2 UDataInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF the receiving of *transparent* data on the B-channel. Data are received without any protocol or control as byte stream. The IdleFlag parameter provided as the default padding character in the Attribute Set is not extracted from the data received.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the Control Plane connection.

Remark: Received data are always provided, but not as a parameter of the message.

Data are provided in the data buffer. This buffer, in this case, shall be mandatory.

Related: UErrorInd.

5.6.2.1.2.3 UErrorInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF that an error has occurred.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the Control Plane connection.
Cause	M	Identifies type of error.

Related: None.

5.6.2.1.3 Messages parameters

This subclause describes parameters for the plane. Table 19 summarises the used parameters.

Table 19: Overview of user parameters

Param. Identif.	Parameter Name	Used in user messages	Used in UAttributeSet	Other use
35	IdleFlag		X	
50	NCOType			X
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
68	Cause	X		

5.6.2.1.3.1 IdleFlag

Description: Flag byte to be sent by the NAF when the user access is idling.

Type: 35.

Fields	Field type	Direction	Required	Comment
IdleFlag	Octet	P	M	Flag byte.

5.6.2.1.3.2 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and transparent user access.

5.6.2.1.3.3 UProtocol

Description: This is used to select the User Plane protocol. The first byte contains the layer 3 protocol requested, the second contains the layer 2 protocol requested and the third contains the layer 1 protocol requested.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	NULL (8)
L1Protocol	Octet	P	M	Transparent B-channel access (1)

Remark: Other values (for other protocols) are provided in subclause 5.6.

5.6.2.1.3.4 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.2.1.3.5 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3).

5.6.2.1.3.6 Cause

Description: This parameter is used to pass Cause information for disconnection to the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	N	M	210 - Overflow.

5.6.2.1.4 State diagram

User messages do not change the state of the connection.

5.6.2.1.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to the transparent B-channel access.

5.6.2.1.6 Selection criteria

No specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.2.1.7 Specific error handling

Errors are dealt with in the following manner: in case of overflow of Incoming Data, PUF shall be sent UErrorInd.

5.6.2.1.8 Static attributes

5.6.2.1.8.1 AttributeSet parameters

Table 20: User Plane Attribute Set (UAttributeSet) Parameters

Parameters	Required	Comment
IdleFlag	C	Flag byte to be sent while idle. See subclause 5.6.2.1.3.1.
UProtocol	O	See 5.6.2.1.3.3.

Remark: These parameters can only be used during NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted, defaults shall be used. The default values are described in annex E.

5.6.2.1.8.2 Static attribute content

Name	: U_TRANSPARENT
UProtocol	: 64 kbit/s
IdleFlag	: 0xFF

5.6.3 Layer 2 Protocols

This subclause describes the specific elements (messages, parameters, attribute sets,...) relating to the layer 2 user protocols. It covers ISO 7776 [4], HDLC with or without error indication, PPP, SDLC and CCITT Recommendation V.110 [17] user protocols. Other layer user protocols are for further study.

5.6.3.1 ISO 7776 protocol

5.6.3.1.1 Introduction

This subclause deals with the ISO 7776 [4] protocol.

The User Plane provides the services for the ISO 7776 [4] protocol using the User Plane protocols on a connection on B-channel. For this access, the NAF considers layer 3 as a Null, as shown in figure 15.

The OSI location of the ISO 7776 [4] protocol is shown in figure 15.

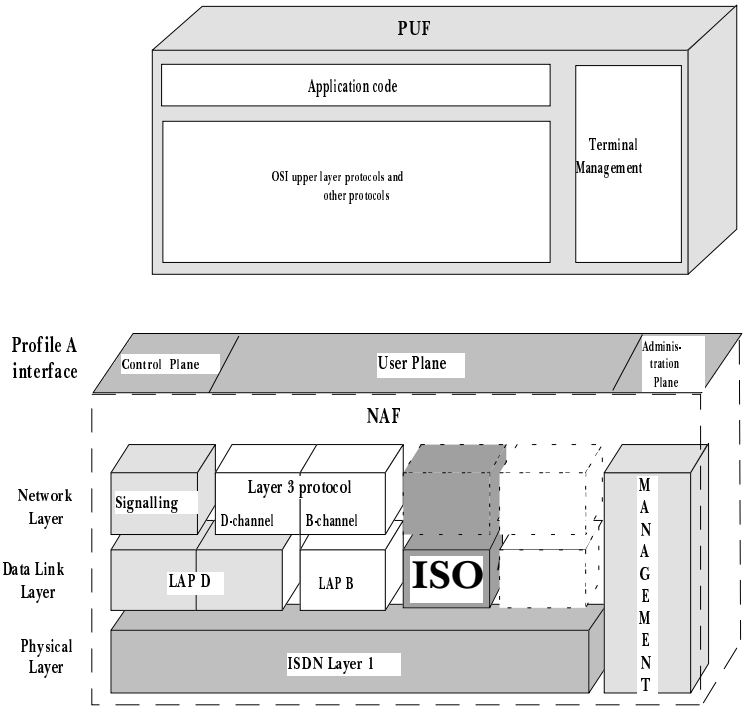


Figure 15: OSI location

The description of the general conventions is provided in subclause 5.3.2.

5.6.3.1.2 Messages

The User Plane messages provide an access to ISO 7776 [4] protocol stacks. The following gives a list and short description of relevant User Plane messages. Table 21 gives an overview of these messages.

Table 21: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
301	1	UConnectReq	Request establishment of a user connection.
302	1	UConnectInd	Indicate establishment of a user connection has been requested.
303	1	UConnectRsp	Indicate acceptance of user connection establishment.
304	1	UConnectCnf	Confirm user connection has been established.
305	1	UDisconnectReq	Request removal of user connection.
306	1	UDisconnectInd	Indicate removal of user connection.
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.
317	1	UReadyToReceiveReq	Used to perform flow control for a user connection.
318	1	UReadyToReceiveInd	Used to indicate flow control status on a user connection.

5.6.3.1.2.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectCnf.

5.6.3.1.2.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectRsp.

5.6.3.1.2.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectInd.

5.6.3.1.2.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF upon the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectReq.

5.6.3.1.2.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: None.

5.6.3.1.2.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
Origin	M	Identifies the initiator of the user connection removal.
Cause	M	Identifies the reason of the user connection removal.

Related: None.

5.6.3.1.2.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the layer 2 data packet size defined at the NCO creation time.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: None.

5.6.3.1.2.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size described at the NCO creation time.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Remark: Data received is always provided, but not as a parameter of the message. Data shall be provided in the data buffer. This buffer, in this case, is mandatory.

Related: None.

5.6.3.1.3 Messages parameters

This subclause describes parameters for the ISO 7776 [4] protocol. Table 22 summarises the used parameters. They are alphabetically ordered.

Table 22: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
38	L2ConnectionMode		X	
39	L2FrameSize		X	
40	L2WindowSize		X	
41	L2XID		X	
50	NCOType			X
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
68	Cause	X		
69	Origin	X		

5.6.3.1.3.1 L2ConnectionMode

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer connection mode to the NAF.

Type: 38.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	dte (1) - Act as a secondary link station (non negotiable). dce (2) - Act as primary link station (non negotiable). auto (3) - Link station role is negotiable by XID exchange.

5.6.3.1.3.2 L2FrameSize

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer 2 frame size to the NAF.

Type: 39.

Fields	Field type	Direction	Required	Comment
Value	Octet string	P	M	Frame size (in octets). Length is fixed to 2 octets. The first octet contents is the most significant byte.

5.6.3.1.3.3 L2WindowSize

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer 2 window size to the NAF.

Type: 40.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Window size

5.6.3.1.3.4 L2XID

Description: This is used to pass details of the layer 2 XID value and its use. XID information field may include values that override some parameters defined elsewhere.

Type: 41.

Fields	Field type	Direction	Required	Comment
Use	Octet	P	M	send (1) - send XID. match (2) - match XID with XID received. If XID does not match, connection shall not be established.
Value	Octet-string	P	M	XID value [Identifier and signature]. Maximum length is 64 octets.

5.6.3.1.3.5 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and link layer user access.

5.6.3.1.3.6 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	ISO 7776 (0)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.1.3.7 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.1.3.8 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	M	Both (3).

5.6.3.1.3.9 Cause

Description: This parameter is used to pass Cause information for disconnection to the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	N	M	See values in table 23.

5.6.3.1.3.10 Origin

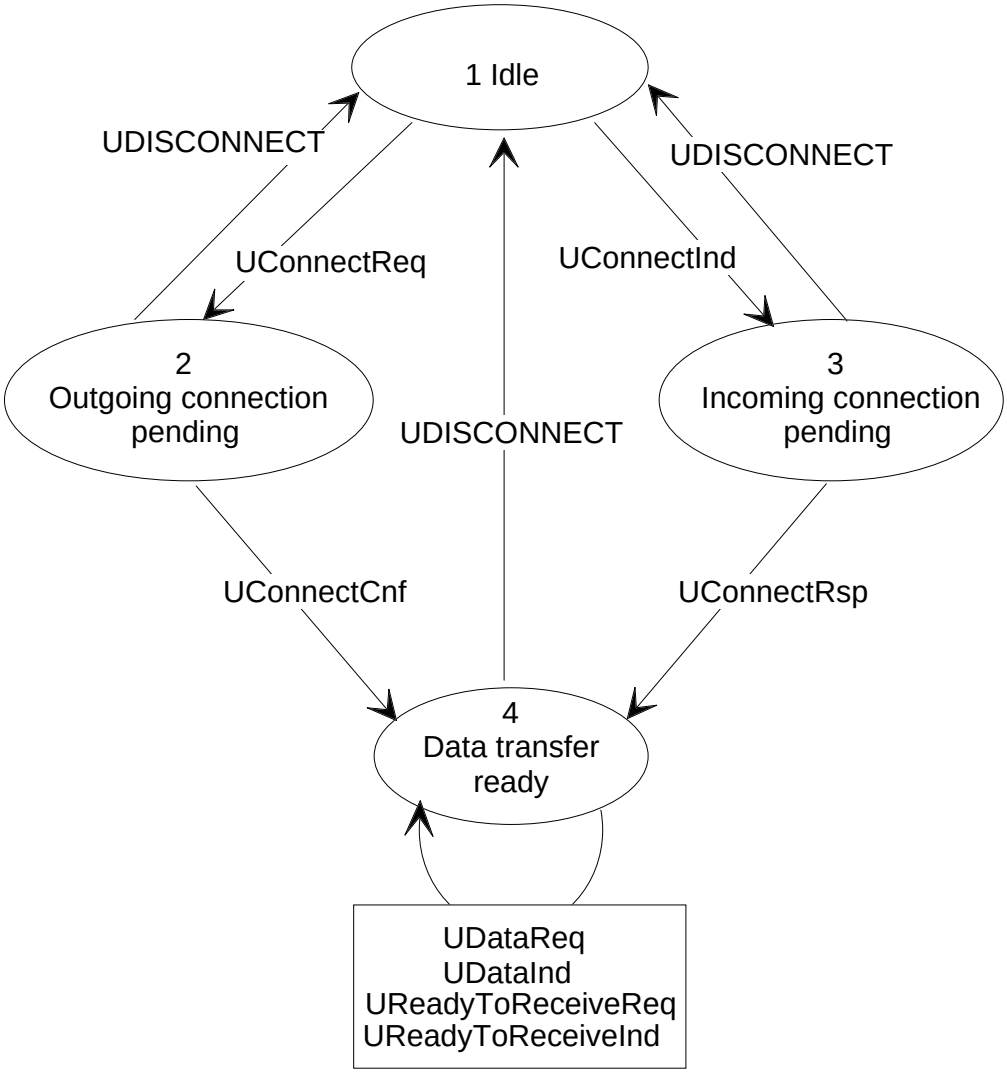
Description: This parameter is used to pass the origin information of the disconnection to the PUF.

Type: 69.

Fields	Field type	Direction	Required	Comment
Value	Octet	N	M	Undefined (1) NAF Provider (2) Remote User (3)

5.6.3.1.4 State diagram

Figure 16 shows the different states of a user connection, using the U-messages, and in which order these messages shall be used.



NOTE: Where UDISCONNECT appears it can be either UDisconnectReq or UDisconnectInd.

Figure 16: Overview of the User Plane messages

5.6.3.1.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to the ISO 7776 [4] protocol.

5.6.3.1.6 Selection criteria

No ISO 7776 [4] protocol specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.3.1.7 Specific error handling and codes

In case of invalid length of UDataReq UserData parameter PUF is sent UDisconnectInd.

The table 23 gives possible values of the Cause parameter.

Table 23: Cause parameter value

Return Code		Meaning	ErrorSpecific Information
Undefined	220	Undefined error situation.	Not present
DiscNorm	241	Disconnection - normal condition.	Not present
InvalidSequence	244	Connection rejected - Invalid sequencing in the frame numbering (transient condition)	Not present
FrameTooBig	245	Connection rejected - Reception of a frame bigger than defined in the NCO value (fixed condition)	Not present

5.6.3.1.8 Static attributes

5.6.3.1.8.1 AttributeSet parameters

Table 24: User Plane Attribute Set (UAttributeSet) Parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.3.1.3.6
L2ConnectionMode	O	See remark. See also subclause 5.6.3.1.3.1
L2FrameSize	O	See remark. See also subclause 5.6.3.1.3.2
L2WindowSize	O	See remark. See also subclause 5.6.3.1.3.3
L2XID	O	See remark. See also subclause 5.6.3.1.3.4

Remark: These parameters can only be used during NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

5.6.3.1.8.2 Static attribute content

```
Name              : U_ISO7776
L2FrameSize       : 128
L2WindowSize      : 7
L2ConnectionMode: Auto
L2XID             : Send and match
```

5.6.3.2 HDLC protocol

5.6.3.2.1 Introduction

This clause deals with the HDLC protocol.

For this access, the NAF considers layer 3 as a Null, as shown in figure 17.

The OSI location of the HDLC protocol is shown in figure 17.

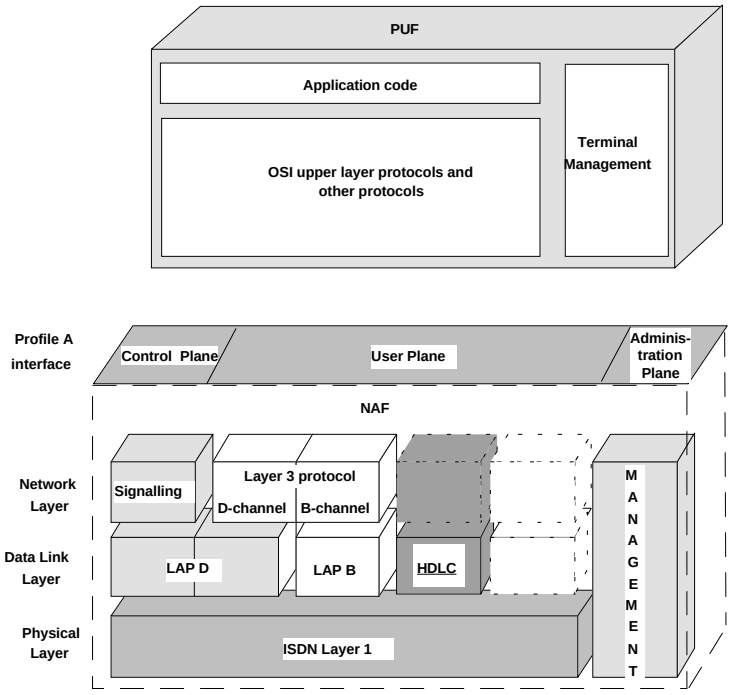


Figure 17: OSI location

A general description of the conventions used is provided in subclause 5.2.3.

5.6.3.2.2 Messages

The User Plane messages provide an access to the protocol stacks. the following is a list and short description of significant User Plane messages. Table 24 gives an overview of these messages.

Table 24: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.

5.6.3.2.2.1 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is limited by the maximum allowed at the Profile A interface i.e. 4 096 octets.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: None.

5.6.3.2.2.2 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is limited by the maximum allowed at the Profile A interface i.e. 4 096 octets.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Remark: Data received is always provided, but not as a parameter of the message. **Data is provided in the data buffer. This buffer, in this case, is mandatory.**

Related: None.

5.6.3.2.3 Messages parameters

This subclause describes parameters for the HDLC plane. Table 25 summarises the used parameters. They are alphabetically ordered.

Table 25: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
50	NCOType			X
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X

5.6.3.2.3.1 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and link layer user access.

5.6.3.2.3.2 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	HDLC (2)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.2.3.3 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.2.3.4 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.3.2.4 State diagram

User messages do not change the state of the connection.

5.6.3.2.5 Co-ordination function

The co-ordination function cannot be used with this User Plane protocol.

5.6.3.2.6 Selection criteria

No specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.3.2.7 Specific error handling and codes

Protocol errors are not available at the interface.

5.6.3.2.8 Static attributes

5.6.3.2.8.1 AttributeSet parameters

Table 26: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.3.2.3.2.

Remark: These parameters can only be used during NCO creation containing Control Plane information. Refer to subclause 5.4.1.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

5.6.3.2.8.2 Static attribute content

```
Name          : U_HDLC
UProtocol     : HDLC
```

5.6.3.3 HDLC protocol with error

5.6.3.3.1 Introduction

This subclause deals with HDLC protocol when errors occur.

For this access, the NAF considers layer 3 as a Null, as shown in figure 18.

The OSI location of the HDLC protocol is shown in figure 18.

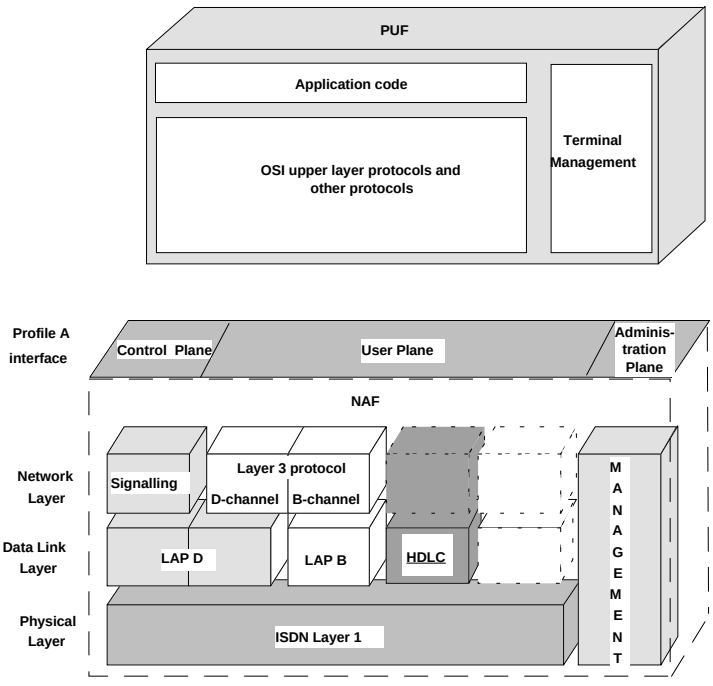


Figure 18: OSI location

General description conventions are provided in subclause 5.2.3.

5.6.3.3.2 Messages

The User Plane messages provide an access to the protocol stacks. Following is a list and short description of significant User Plane messages. Table 27 gives an overview of these messages.

Table 27: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.

5.6.3.3.2.1 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is limited by the maximum allowed at the Profile A interface i.e. 4 096 octets.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: None.

5.6.3.3.2.2 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is limited by the maximum allowed at the Profile A interface i.e. 4 096 octets.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
Cause	O	Identifies the type of error.

Remark: Data received is always provided, but not as a parameter of the message. Data is provided in the data buffer. This buffer, in this case, is mandatory.

Related: None.

5.6.3.3.3 Messages parameters

This subclause describes parameters for the HDLC plane. Table 28 summarises used parameters. They are alphabetically ordered.

Table 28: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
50	NCOType			X
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
68	Cause	X		

5.6.3.3.3.1 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and link layer user access.

5.6.3.3.3.2 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	HDLC protocol with error (3)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.3.3.3 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.3.3.4 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.3.3.3.5 Cause

Description: This parameter is used to pass Cause information to/from the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	210 - Overflow 211 - Framing error

5.6.3.3.3.6 State diagram

User messages do not change the state of the connection.

5.6.3.3.4 Co-ordination function

The co-ordination function cannot be used with this User Plane protocol.

5.6.3.3.5 Selection criteria

No specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.3.3.6 Specific error handling

In the case of invalid length of UDataReq UserData parameter, PUF is sent UDataInd with Cause parameter.

5.6.3.3.7 Static attributes

5.6.3.3.7.1 AttributeSet parameters

Table 29: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.3.3.3.2.

Remark: These parameters can only be used during NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

5.6.3.3.7.2 Static attribute content

Name	: U_HDLC_E
UProtocol	: HDLC

5.6.3.4 PPP protocol

5.6.3.4.1 Introduction

This subclause deals with the point-to point protocol (PPP).

The User Plane provides the services for PPP using the User Plane protocols on a connection over the B-channel. For this access, the NAF considers layer 3 as a Null, as shown in figure 19.

The OSI location of the PPP is shown in figure 19.

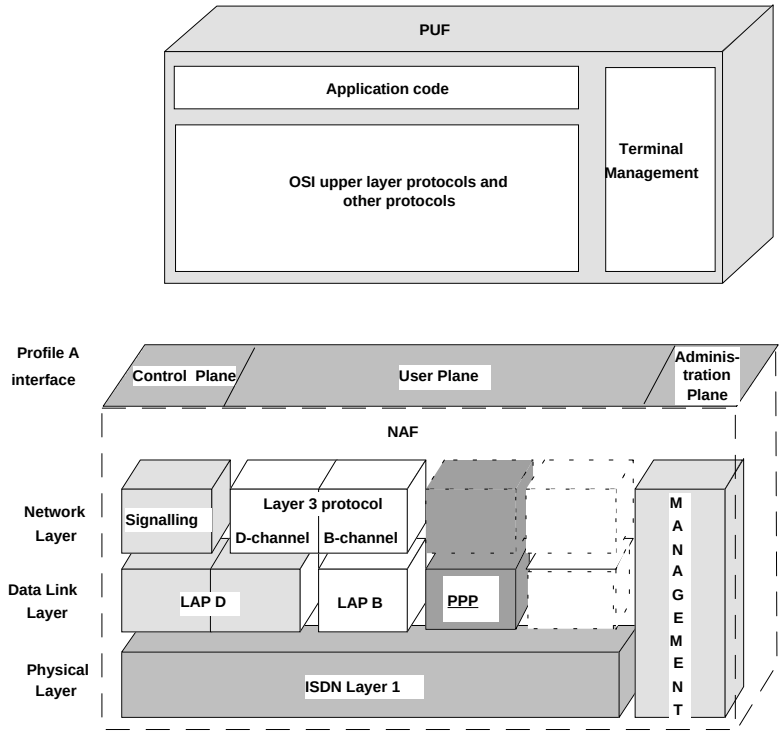


Figure 19: OSI location

NOTE: PPP is defined as a set of protocols which can be divided into groups:

- Link Control Protocols (LCP) in charge of the establishment, configuration and testing of the data link connection;

- a family of Network Control Protocol (NCP) in charge of the establishment and configuration of the different network layers protocols.

The PPP implementation in Profile A applies only on the Link Control Protocol LCP (RFC 1661 [10]), PPP Link Quality Monitoring (RFC 1333) and PPP Authentification Protocols (RFC 1334).
The NCPs which handle problems concerning the configuration of network protocols are defined in specific documents. These are not covered by the specifications.

General description of the conventions use is provided in subclause 5.2.3.

5.6.3.4.2 Messages

The User Plane messages provide an access to PPP stacks. The following is a list and short description of relevant User Plane messages. Table 30 gives an overview of these messages.

Table 30: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
301	1	UConnectReq	Request establishment of a user connection.
302	1	UConnectInd	Indicate establishment of a user connection has been requested.
303	1	UConnectRsp	Indicate acceptance of user connection establishment.
304	1	UConnectCnf	Confirm user connection has been established.
305	1	UDisconnectReq	Request removal of user connection.
306	1	UDisconnectInd	Indicate removal of user connection.
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.
319	1	UErrorInd	Indicate an error.

5.6.3.4.2.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
PPPNegotiation	M	Indicates the requested value.

Related: UConnectCnf.

5.6.3.4.2.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
PPPNegotiation	M	Indicates the value proposed for this user connection.

Related: UConnectRsp.

5.6.3.4.2.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectInd.

5.6.3.4.2.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF on the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectReq.

5.6.3.4.2.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
PPPCause	O	PPP reason to remove the user connection.

Related: None.

5.6.3.4.2.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
PPPOrigin	M	Identifies the initiator of the user connection removal.
PPPCause	O	PPP reason to remove the user connection.
PPPDiagnostic	C	Complementary information for PPPCause. Optional if PPPCause parameter supplied.

Related: None.

5.6.3.4.2.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Address field is set to "11111111" (All-Station address) and control field is set to "00000011"(Unnumbered Information) with bit P/F set to zero. The FCS is inserted at the end of each data block with the flag, transparently by the NAF.

Related: None.

5.6.3.4.2.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Remark: Data received is always provided, but not as a parameter of the message. Data is provided in the data buffer. This buffer, in this case, is mandatory.

Address field is set to "11111111" (All-Station address) and control field is set to "00000011"(Unnumbered Information) with bit P/F set to zero. The FCS is inserted at the end of each data block with the flag, transparently by the NAF.

Related: None.

5.6.3.4.2.9 UErrorInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF that an error has occurred.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the Control Plane connection.
PPPCause	M	Identifies type of error.

Related: None.

5.6.3.4.3 Messages parameters

This subclause describes parameters for the PPP plane. Table 31 summarises user parameters.

Table 31: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
50	NCOType			X
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
69	PPPOrigin	X		
70	PPPCause	X		
71	PPPDiagnostic	X		
74	PPPNegotiation	X	X	

5.6.3.4.3.1 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and link layer user access.

5.6.3.4.3.2 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	PPP (4)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.4.3.3 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.4.3.4 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.3.4.3.5 PPPCause

Description: This parameter is used to pass PPPCause information to/from the PUF.

Type: 70.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See values in table 32.

5.6.3.4.3.6 PPPDiagnostic

Description: This parameter is used to pass PPP Diagnostic information associated to a PPP Cause.

Type: 71.

Fields	Field type	Direction	Required	Comment
DiagType	Octet	N	M	Indicate the type of diagnostic associated with the PPPCause .ConfNoConverging (0)
NoConvergingDiag	Octet-string	N	M	Diagnostic associated with ConfNoConverging (note)
NOTE: These elements are ordered in the same way that they have been defined in the PPPNegotiation message (see PPPNegotiation). Furthermore, the bits corresponding to the non-acknowledged options are set and those corresponding to the acknowledged options are reset.				

5.6.3.4.3.7 PPPNegotiation

Description: This parameter is used to indicate the PPP negotiation to perform.

Type: 74.

Fields		Field type	Direction	Required	Comment
PPPNegociation	Usage	Octet-string	B	M	Indicates if the following values are included. Length is fixed to 2 octets (note 1)
	MRUlocal	Octet-string	B	C	Indicates the size of the Maximum Receive Unit of the local peer Default (0) Length is fixed to 2 (note 2)
	MRUremote	Octet-string	N	C	Indicates the size of the Maximum Receive Unit of the remote peer Length is fixed to 2 octets (note 2)
	Authentproto	Octet	B	C	Indicate the type of the authentication to perform. These values are exclusive. Default (0) PAP (1) CHAP (2) (note 2)
	Qualityproto	Octet-string	B	C	Indicate the value of the reporting period for quality protocol Default (0) Length is fixed to 4 octets (note 2)
	Magicnumber	Octet-string	B	C	Indicate the value of the magic number to use Default (0) Length is fixed to 4 (note 2)

(continued)

Fields		Field type	Direction	Required	Comment
	Protocolcomp	Octet	B	C	Indicate if the protocol field compression is to be set (note 2).
	Addresscomp	Octet	B	C	Indicate if the address field compression is to be set (note 2).
	FCSAlternatives	Octet-string	B	C	Indicate the value of the FCS format to use Default (0) Length is fixed to 4 octets (note 2)
	SelfDescPadding	Octet-string	B	C	Indicate the value of the Self Describing Padding to use Default (0) Length is fixed to 4 octets
	CallBack	Octet	B	C	Indicate if the callback option is to be set (note 2).
	CompoundFrame	Octet	B	C	Indicate if the CompoundFrame option is to be set (note 2).
NOTE 1:	Each bit of the indicator corresponds to an option ordered as indicated in the array (that means the first bit refers to the MRUlocal option, the second to the MRUremote option, the third to *Authentproto* option and so on). When the indicator concerning an option is not set, it means that the PPP option has not to be negotiated.				
NOTE 2:	Before defining a negotiation parameter, the PUF has to check if the functionality is served by the NAF by using the PciGetProperty. See subclause 9.4.9.				

5.6.3.4.3.8 PPPOrigin

Description: This parameter is used to pass PPP origin information to/from the PUF.

Type: 69.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	undefined (1) NAF Provider (2) PUF User (3)

5.6.3.4.4 State diagram

Figure 20 shows the different states of a user connection, using the U-messages, and in which order these messages shall be used.

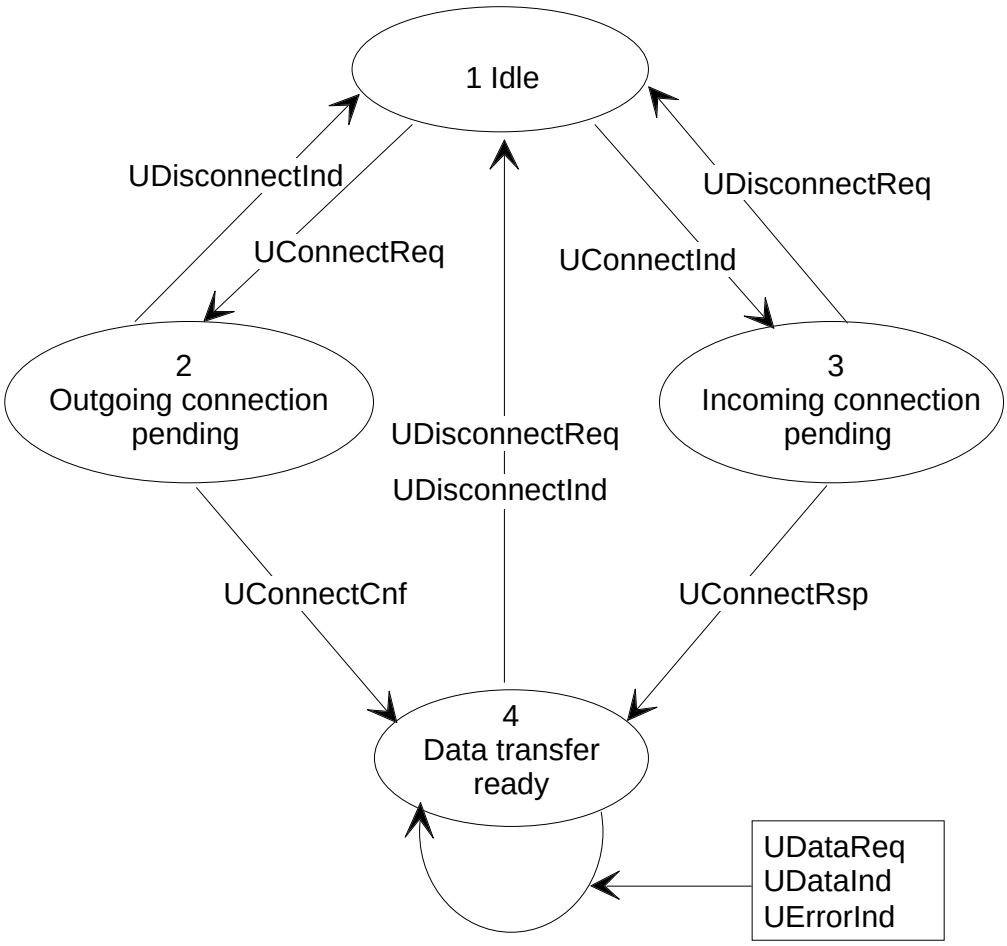


Figure 20: Overview of the User Plane messages

5.6.3.4.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to PPP.

5.6.3.4.6 Selection criteria

No PPP specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.3.4.7 Specific error handling and codes

Errors are dealt with in the following manner:

5.6.3.4.7.1 Errors

In case of protocol reject information from the remote part, the PUF is sent UErrorInd.
In case of invalid length of UDataReq UserData parameter, data is ignored.

5.6.3.4.7.2 Causes

These values can be specified and are returned in the PPPCause parameter.

Table 32: PPPCause parameter value

Return Code		Meaning	ErrorSpecific Information
Undefined	220	Undefined error situation.	Not present
DiscNorm	241	Disconnection - normal condition.	Not present
ConfNoConverging	244	Connection rejected - host not responding (transient condition)	Not present
Hostunreachable	245	Connection rejected - configurations cannot match (fixed condition)	Not present
Protocol Error	212	Protocol Error	Not present

5.6.3.4.8 Static attributes

5.6.3.4.8.1 AttributeSet parameters

Table 33: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.3.4.3.2.
MRUlocal	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
MRUremote	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
Authentproto	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
Qualityproto	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
MagicNumber	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
Protocolcomp	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
Addresscomp	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
FCSAlternatives	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
SelfDescPadding	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
CallBack	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).
CompoundFrame	O	See remark. See also subclause 5.6.3.4.3.7 (PPPNegotiation).

Remark: These parameters can only be used during a NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

5.6.3.4.8.2 Static attribute content

Name	: U_PPP
UProtocol	: PPP
MRUlocal	: 1500
MRUremote	: 1500
Authentproto	: none
Qualityproto	: none
MagicNumber	: none
Protocolcomp	: none
AddressComp	: none
FCSAlternatives	: none
SelfDescPadding	: none
CallBack	: none
CompoundFrame	: none

5.6.3.4.9 Protocol specific NAF property information

The PPP specific parameters of NAF-Property are shown in table 34.

Table 34: TLV coded NAF-Property parameter

Parameter	Provided	TLV Coding			Comment and values
		TypeID	Length	Value	
PPPNegotiation	M	14	2..27	Octet	Indicates the PPP options provided by the NAF

See also the PciGetProperty function in subclause 5.3.1.3.

5.6.3.5 SDLC protocol

5.6.3.5.1 Introduction

This clause deals with the SDLC protocol.

The User Plane provides the services for SDLC using the User Plane protocols on a connection over a B-channel. For this access, the NAF considers layer 3 as a Null, as shown in figure 21.

SDLC protocol supported is an SDLC link in normal response mode having a point-to-point configuration. Overview and protocol information is provided in IBM publication "IBM Synchronous Data Link Control Concepts" (GA27-3093).

The OSI location of the SDLC protocol is shown in figure 21.

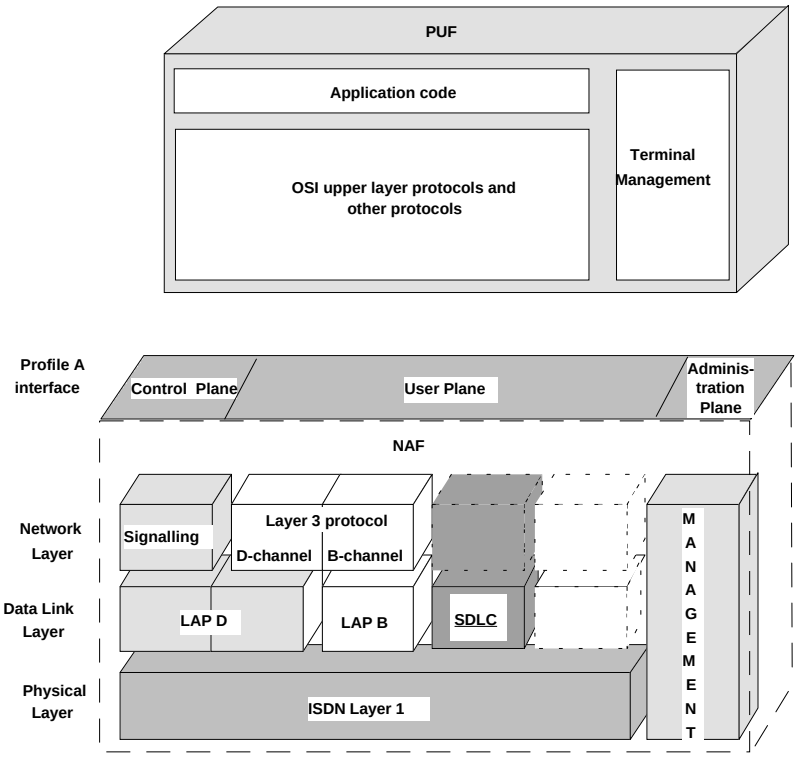


Figure 21: OSI location

General description of the conventions used are provided in subclause 5.2.3.

5.6.3.5.2 Messages

The User Plane messages provide an access to SDLC protocol stacks. Table 35 provides a list and short description of relevant User Plane messages.

Table 35: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
301	1	UConnectReq	Request establishment of a user connection.
302	1	UConnectInd	Indicate establishment of a user connection has been requested.
303	1	UConnectRsp	Indicate acceptance of user connection establishment.
304	1	UConnectCnf	Confirm user connection has been established.
305	1	UDisconnectReq	Request removal of user connection.
306	1	UDisconnectInd	Indicate removal of user connection.
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.
309	1	UExpeditedDataReq	Request expedited data transfer on an established user connection.
310	1	UExpeditedDataInd	Indicate presence of transferred expedited data on an established user connection.
317	1	UReadyToReceiveReq	Used to perform flow control for a user connection.
318	1	UReadyToReceiveInd	Used to indicate flow control status on a user connection.

5.6.3.5.2.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectCnf.

5.6.3.5.2.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a user connection. This message informs the PUF of the end of transient idle state of the user connection caused by Data Link Layer resetting.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectRsp.

5.6.3.5.2.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectInd.

5.6.3.5.2.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF upon the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectReq.

5.6.3.5.2.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: None.

5.6.3.5.2.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed. This message may inform the PUF of transient idle state of the user connection caused by Data Link Layer reset. In this case, the SDLCCause parameter value is DiscTrans (disconnection - transient condition). See subclause 5.6.3.5.3.11.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
SDLCOrigin	M	Identifies the initiator of the user connection removal.
SDLCCause	O	SDLC reason to remove the user connection.

Related: None.

5.6.3.5.2.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment. No fragmentation mechanism is available at the SDLC link layer level.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: None.

5.6.3.5.2.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment. No fragmentation mechanism is available at the SDLC link layer level.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Remark: Data received is always provided, but not as a parameter of the message. Data is provided in the data buffer. This buffer, in this case, is mandatory.

Related: None.

5.6.3.5.2.9 UExpeditedDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send expedited data. This data is not constrained by the flow control mechanism used to control UDataReq messages. SDLC expedited data is transmitted in an Unnumbered Information frame.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
UserData	M	Expedited data to transfer.

Related: None.

5.6.3.5.2.10 UExpeditedDataInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF the reception of expedited data. This data was not constrained by the flow control mechanisms used to control UDataInd messages. SDLC expedited data is received in an Unnumbered Information frame.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
UserData	M	Expedited data received.

Related: None.

5.6.3.5.2.11 UReadyToReceiveReq

Class: 1 (Basic Class).

Description: This message allows the PUF to indicate to the NAF if it can accept incoming data (UDataInd message). This message can only apply to an already established user connection. Setting the ReadyFlag parameter to TRUE allows the NAF to transfer incoming data to the PUF. Setting the ReadyFlag to FALSE inhibits the transfer.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the PUF is ready to accept incoming data.

Remarks: For a given connection, if more than one message with the same flag value is sent, it shall be ignored by the NAF.

Related: UDataInd.

5.6.3.5.2.12 UReadyToReceiveInd

Class: 1 (Basic Class).

Description: This message allows the NAF to indicate to the PUF if the user connection permits the sending of data (UDataReq messages). This message can only apply to an already established user connection. If the ReadyFlag parameter value is FALSE, the NAF can not send data. If the value is TRUE the NAF indicates that data transfer is allowed.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the NAF is ready to receive data for transmission on a user connection.

Related: UDataReq.

5.6.3.5.3 Messages parameters

This subclause describes parameters for the SDLC plane. Table 36 summarises used parameters.

Table 36: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
38	L2ConnectionMode		X	
39	L2FrameSize		X	
40	L2WindowSize		X	
41	L2XID		X	
50	NCOType			X
55	ReadyFlag	X		
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
65	UserData	X		
68	SDLCCause	X		
69	SDLCOrigin	X		

5.6.3.5.3.1 L2ConnectionMode

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer connection mode to the NAF.

Type: 38.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	dte (1) - Act as a secondary link station (non negotiable). dce (2) - Act as primary link station (non negotiable). auto (3) - Link station role is negotiable by XID exchange.

5.6.3.5.3.2 L2FrameSize

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer 2 frame size to the NAF.

Type: 39.

Fields	Field type	Direction	Required	Comment
Value	Octet string	P	M	Frame size (in octets). Length is fixed to 2 octets. The first octet contents is the most significant byte.

5.6.3.5.3.3 L2WindowSize

Description: This parameter is used only if it is not defined in L2XID value field. It is used to pass details of the layer 2 window size to the NAF.

Type: 40.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Window size

5.6.3.5.3.4 L2XID

Description: This is used to pass details of the layer 2 XID value and its use. The XID information field may include values that override some parameters defined elsewhere. DLC XID information field formats are described in IBM publication "Systems Network Architecture - Formats" (GA27-3136-11).

Type: 41.

Fields	Field type	Direction	Required	Comment
Use	Octet	P	M	Not relevant for SDLC protocol.
Value	Octet-string	P	M	XID value [Identifier and signature]. Formatted DLC XID information field for SDLC protocol. Maximum length is 127.

5.6.3.5.3.5 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and link layer user access.

Remark: An SDLC connection can only be defined by a C/U type NCO. No U3G type NCO can be grouped to an NCO defining an SDLC connection.

5.6.3.5.3.6 ReadyFlag

Description: This parameter is used to request and indicate flow control status on a user connection.

Type: 55.

Fields	Field type	Direction	Required	Comment
Usage	Boolean	B	M	TRUE - Data transfer is allowed. FALSE - Data transfer is not allowed.

5.6.3.5.3.7 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	SDLC (5)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.5.3.8 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.5.3.9 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.3.5.3.10 UserData

Description: This parameter is used to pass data that is limited in size to/from the PUF.

Type: 65.

Fields	Field type	Direction	Required	Comment
Data	Octet-string	B	M	128 octets is the maximum size.

5.6.3.5.3.11 SDLCCause

Description: This parameter is used to pass SDLC Cause information to/from the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See values in table 36.

5.6.3.5.3.12 SDLCOrigin

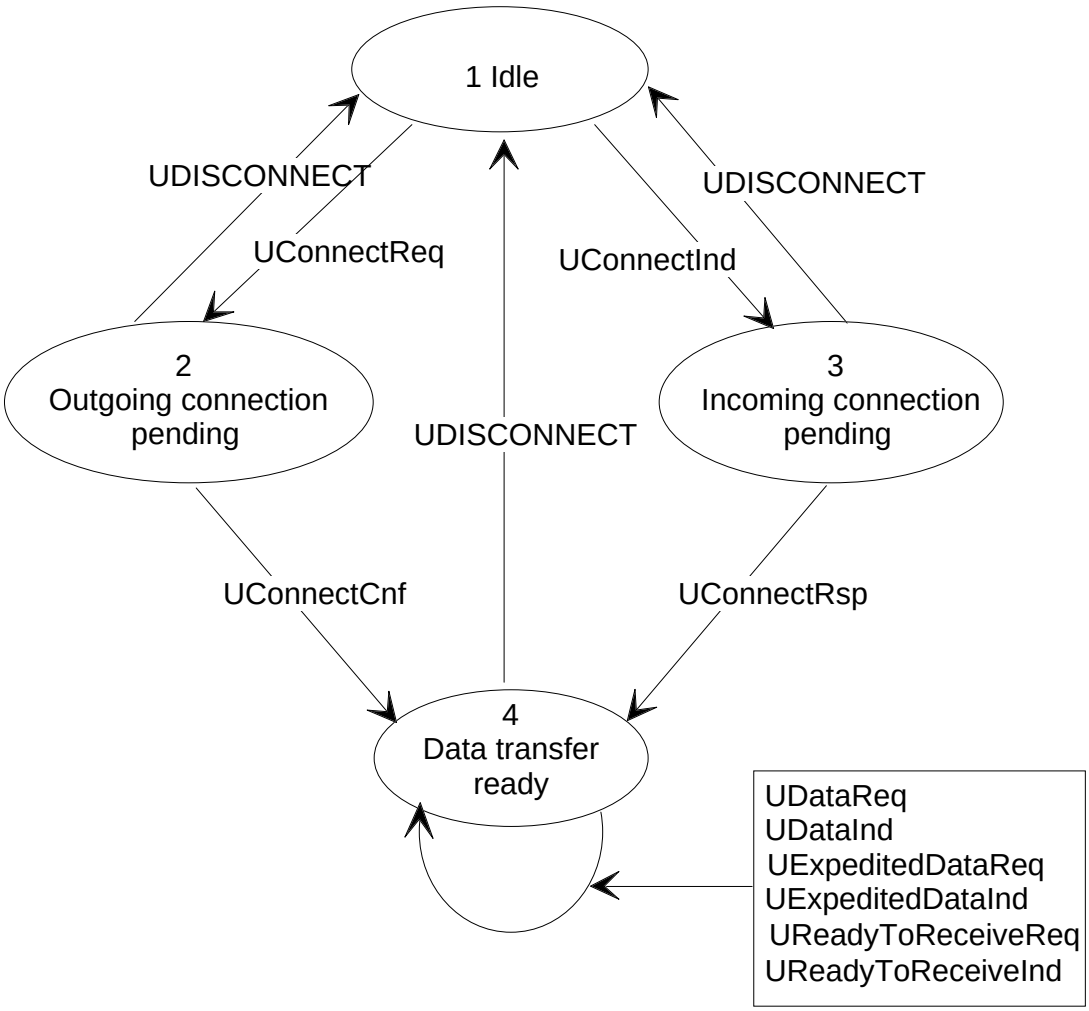
Description: This parameter is used to pass SDLC origin information to/from the PUF.

Type: 69.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	undefined (1) NAF Provider (2) PUF User (3)

5.6.3.5.4 State diagram

Figure 22 shows the different states of a user connection, using the U-messages, and in which order these messages shall be used.



NOTE: Where UDISCONNECT appears it can be either UDisconnectReq or UDisconnectInd.

Figure 22: Overview of the User Plane messages

5.6.3.5.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to SDLC.

5.6.3.5.6 Selection criteria

No SDLC specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.3.5.7 Specific error handling and codes

Errors are dealt with as stated in subclauses 5.6.3.5.7.1 and 5.6.3.5.7.2.

5.6.3.5.7.1 Invalid use of user messages

In case of:

- invalid length of UDataReq UserData parameter;
- invalid use of ExpeditedData,

action is:

- PUF is sent UDisconnectInd.

5.6.3.5.7.2 Causes

These values are specified in table 37 and are returned in the SDLCCause parameter.

Table 37: SDLCCause parameter value

Return Code		Meaning	ErrorSpecific Information
Undefined	220	Undefined error situation.	Not present
DiscTrans	225	Disconnection - transient condition. Indicates that Data Link layer is resetting.	Not present
DiscPerm	226	Disconnection - permanent condition. Indicates that the remote station is no longer reachable.	Not present
DiscNorm	241	Disconnection - normal condition. Indicates that the disconnection has been requested by the remote station.	Not present
ConRejectTrans	244	Connection rejection - transient condition. Indicates that Data Link layer activation has been denied by the remote station.	Not present
ConRejectPerm	245	Connection rejection - fixed condition. Indicates that the remote station is unreachable.	Not present

5.6.3.5.8 Static attributes

5.6.3.5.8.1 AttributeSet parameters

Table 38: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.3.5.3.7.
L2ConnectionMode	O	See remark. See also subclause 5.6.3.5.3.1
L2WindowSize	O	See remark. See also subclause 5.6.3.5.3.3.
L2FrameSize	O	See remark. See also subclause 5.6.3.5.3.2
L2XID	O	See remark. See also subclause 5.6.3.5.3.4

Remark: These parameters can only be used during an NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for more details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

L2ConnectionMode, L2WindowSize and L2FrameSize parameters may contain user defined values or default configuration values when not provided by the NAF, or may contain values resulting from XID exchange negotiation. L2XID parameter contains the DLC XID information field.

5.6.3.5.8.2 Static attribute content

```
Name              : U_SDLC
UProtocol         : SDLC
L2ConnectionMode: dte
L2WindowSize      : 7
L2FrameSize       : 265
L2XID             : Not used
```

5.6.3.6 V.110 protocol

5.6.3.6.1 Introduction

This subclause deals with the CCITT Recommendation V.110 [17] protocol. This protocol allows a PUF to request a NAF provide a B-channel running the CCITT Recommendation V.110 [17] protocol. Both synchronous and asynchronous options of CCITT Recommendation V.110 [17] are allowed.

This protocol uses NULL layer 3 protocol as shown in figure 23.

The OSI location of the CCITT Recommendation V.110 [17] protocol is shown in figure 23.

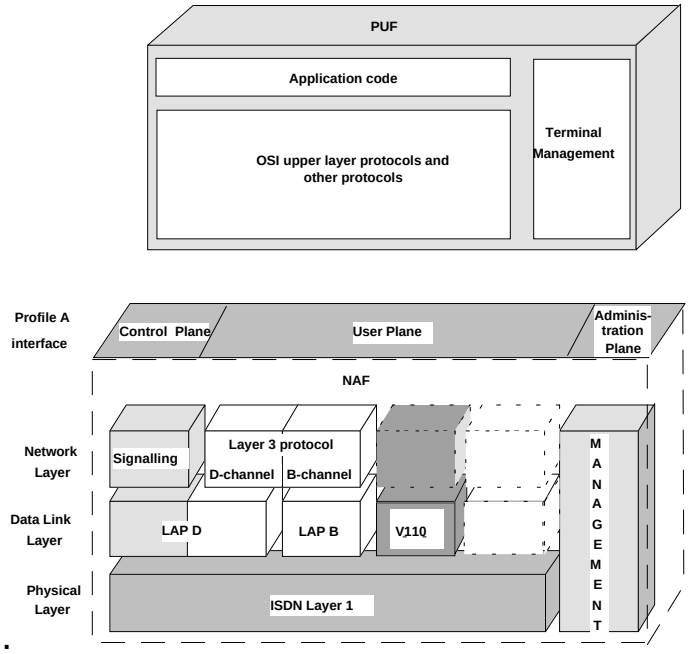


Figure 23: OSI location

A general description of the conventions used is provided in subclause 5.2.3.

A CCITT Recommendation V.110 [17] negotiation may occur via the parameter "BearerCap". In this case, octets 5a, 5b, 5c and 5d of the parameter may be concerned (see subclause 5.5 and 5.7).

5.6.3.6.2 Messages

The User Plane messages provide an access to CCITT Recommendation V.110 [17] protocol stacks. the following gives a list and short description of relevant User Plane messages. Table 39 gives an overview of these messages.

Table 39: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
301	1	UConnectReq	Request establishment of a user connection.
302	1	UConnectInd	Indicate establishment of a user connection has been requested.
303	1	UConnectRsp	Indicate acceptance of user connection establishment.
304	1	UConnectCnf	Confirm user connection has been established.
305	1	UDisconnectReq	Request removal of user connection.
306	1	UDisconnectInd	Indicate removal of user connection.
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.
317	1	UReadyToReceiveReq	Used to perform flow control for a user connection.
318	1	UReadyToReceiveInd	Used to indicate flow control status on a user connection.

5.6.3.6.2.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectCnf.

5.6.3.6.2.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectRsp.

5.6.3.6.2.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectInd.

5.6.3.6.2.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF on the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectReq.

5.6.3.6.2.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: None.

5.6.3.6.2.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
V.110Origin	M	Identifies the initiator of the user connection removal.
V.110Cause	O	V.110 reason to remove the user connection.

Related: None.

5.6.3.6.2.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is limited by the maximum allowed in Profile A i.e. 4 096 octets.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: None.

5.6.3.6.2.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is limited by the maximum allowed in Profile A i.e. 4 096 octets.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Remark: Data received is always provided, but not as a parameter of the message. Data is provided in the data buffer. This buffer, in this case, is mandatory.

Related: None.

5.6.3.6.2.9 UReadyToReceiveReq

Class: 1 (Basic Class).

Description: This message allows the PUF to indicate to the NAF if it can accept incoming data (UDataInd message). This message can only apply to an already established user connection. Setting the ReadyFlag parameter to TRUE allows the NAF to transfer incoming data to the PUF. Setting the ReadyFlag to FALSE inhibits the transfer.

This flow control mechanism does not imply end-to-end flow control.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the PUF is ready to accept incoming data.

Remarks: For a given connection, if more than one message with the same flag value is sent, it shall be ignored by the NAF.

Related: UDataInd.

5.6.3.6.2.10 UReadyToReceiveInd

Class: 1 (Basic Class).

Description: This message allows the NAF to indicate to the PUF if the user connection permits the sending of data (UDataReq messages). This message can only apply to an already established user connection. If the ReadyFlag parameter value is FALSE, the NAF can not send data. If the value is TRUE the NAF indicates that data transfer is allowed.
This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the NAF is ready to receive data for transmission on a user connection.

Related: UDataReq.

5.6.3.6.3 Messages parameters

This subclause describes parameters for the CCITT Recommendation V.110 [17] User Plane protocol.

Table 40: Overview of user parameters

Identif.	Parameter Name	Used in user messages	Used in UAttributeSet	Other use
50	NCOType			X
55	ReadyFlag	X		
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
68	V.110Cause	X		
69	V.110Origin	X		
75	FlowControlMechanism		X	
76	FlowControlCharacters		X	
77	MomentNumber		X	
78	V.110BChannelDisconnection		X	

5.6.3.6.3.1 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	C/U (3) - signalling and transparent user access.

5.6.3.6.3.2 ReadyFlag

Description: This parameter is used to request and indicate flow control status on a user connection.

Type: 55.

Fields	Field type	Direction	Required	Comment
Usage	Boolean	B	M	TRUE - Data transfer is allowed. FALSE - Data transfer is not allowed.

5.6.3.6.3.3 UProtocol

Description: This is used to select the User Plane protocol. If the length is 3, the first octet contains the layer 3 protocol requested, the second octet contains the layer 2 protocol requested and the third octet contains the layer 1 protocol requested.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4)
L2Protocol	Octet	P	M	V.110 asynchronous (6) V.110 synchronous (7)
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.3.6.3.4 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.3.6.3.5 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.3.6.3.6 V.110Cause

Description: This parameter is used to pass CCITT Recommendation V.110 [17] cause information to/from PUF.

Type: 67.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See values in table 40.

5.6.3.6.3.7 V.110Origin

Description: This parameter is used to pass CCITT Recommendation V.110 [17] origin information to/from PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	undefined (1) NAF Provider (2) PUF User (3)

5.6.3.6.3.8 FlowControlMechanism

Description: This parameter is used to negotiate the flow control mechanism for a CCITT Recommendation V.110 [17] connection. Two possibilities exist: first is the XON/XOFF characters, second is via V.24 105/106 signals. This parameter is used for end-to-end negotiation.

Type: 75.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Type of mechanism to use: 0 XON/XOFF characters 1 105/106 signal Default value is 0 (XON/XOFF).

5.6.3.6.3.9 FlowControlCharacters

Description: This parameter is used to set the characters to define flow control characters for a CCITT Recommendation V.110 [17] connection. The characters may be different for each direction, so two characters are mandatory to be provided, even if they have the same value. This parameter only has a local meaning.

Type: 76.

Fields	Field type	Direction	Required	Comment
Value	Octet String	P	M	Fixed length is 2. The first character identifies the XON character. Default value is 16. The second character identifies the XOFF character. Default value is 18.

5.6.3.6.3.10 MomentNumber

Description: This parameter is used to set the number of moments for a CCITT Recommendation V.110 [17] connection. It only has a local meaning.

Type: 77.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Number of moments.

5.6.3.6.3.11 V.110BChannelDisconnection

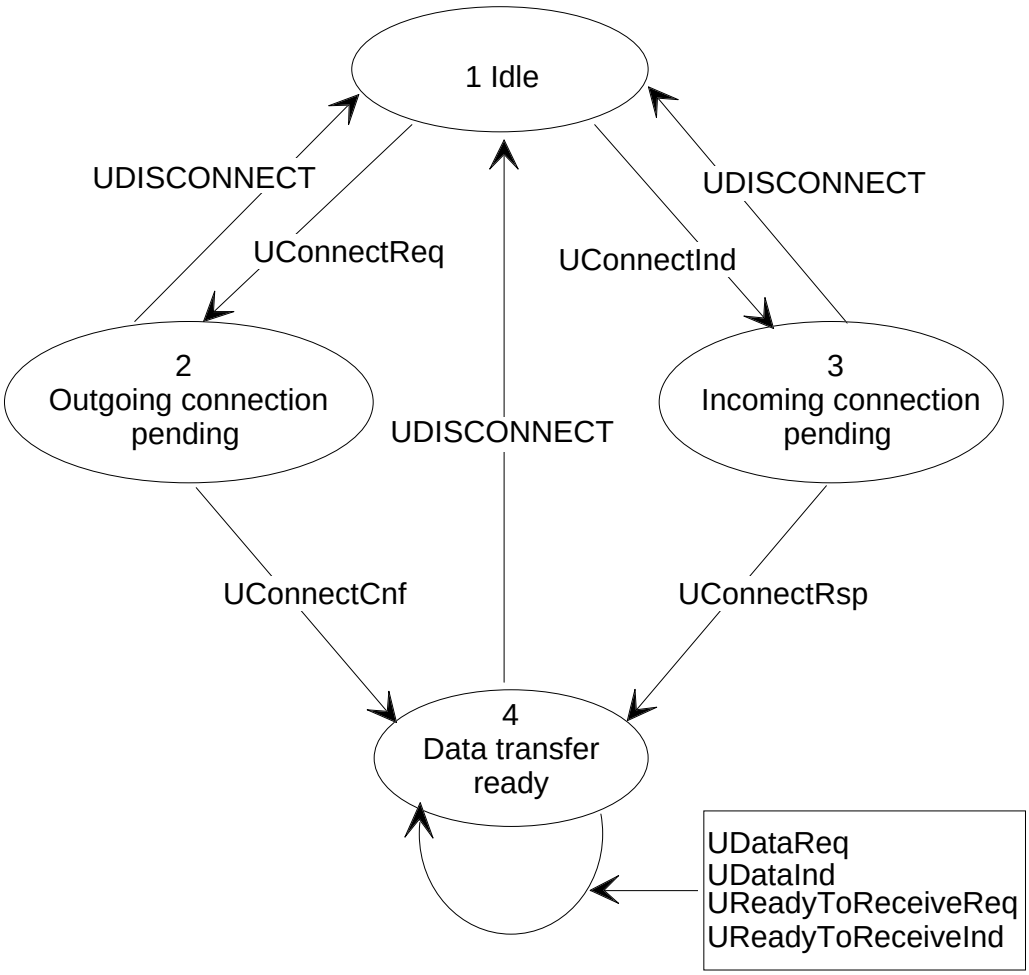
Description: A CCITT Recommendation V.110 [17] disconnection may imply the B-channel disconnection. This parameter is used to set this information. It only has a local meaning.

Type: 78.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	V.110 implies B-channel disconnection: 0 No disconnection 1 Disconnection

5.6.3.6.4 State diagram

Figure 24 shows the different states of a user connection, using the U-messages, and in which order these messages shall be used.



NOTE: Where UDISCONNECT appears it can be either UDisconnectReq or UDisconnectInd.

Figure 24: Overview of the User Plane messages

5.6.3.6.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to CCITT Recommendation V.110 [17] access.

5.6.3.6.6 Selection criteria

No specific parameters are to be used. General NCO criteria are provided in subclause 5.8.

5.6.3.6.7 Specific error handling and codes

Errors are dealt with as given in subclauses 5.6.3.6.7.1 and 5.6.3.6.7.2.

5.6.3.6.7.1 Invalid use of User Plane messages

In the case of invalid length of UDataReq UserData parameter, PUF is sent UDisconnectInd.

5.6.3.6.7.2 Causes

These values can be specified and are returned in the V.110Cause parameter.

Table 41: V.110Cause parameter value

Return Code		Meaning	ErrorSpecific Information
Undefined	220	Undefined error situation.	Not present
DiscNorm	241	Disconnection - normal condition.	Not present
ConRejectTrans	244	Connection rejected (transient condition)	Not present
ConRejectPerm	245	Connection rejected (permanent condition)	Not present

5.6.3.6.8 Static attributes

5.6.3.6.8.1 AttributeSet parameters

Table 42: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
UProtocol	O	See subclause 5.6.3.6.3.3.
FlowControlMechanism	O	See subclause 5.6.3.6.3.8.
FlowControlCharacters	O	See subclause 5.6.3.6.3.9.
MomentNumber	O	See subclause 5.6.3.6.3.10.
V.110BChannelDisconnection	O	See subclause 5.6.3.6.3.11.

Remark: These parameters can only be used during a NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used. The default values described in annex E.

5.6.3.6.8.2 Static attribute content

Name	: U_V.110
UProtocol	: V.110 asynchronous
FlowControlMechanism	: 0
FlowControlCharacters	: 17 19
MomentNumber	: not defined
V.110BChannelDisconnection	: 0

5.6.4 Layer 3 protocols

This subclause describes the specific elements (messages, parameters, attribute sets,...) relating to the layer 3 user protocols. It covers ETS 300 080 [1], ISO 8208 [3] and T.70NL user protocols. Other layer 3 user protocols are for further study.

5.6.4.1 ISO 8208 protocol and ETS 300 080 protocol

5.6.4.1.1 Introduction

This clause deals with the ISO 8208 [3] protocol and with the ETS 300 080 [1] protocol. Figure 25 shows the localisation of the user protocol access.

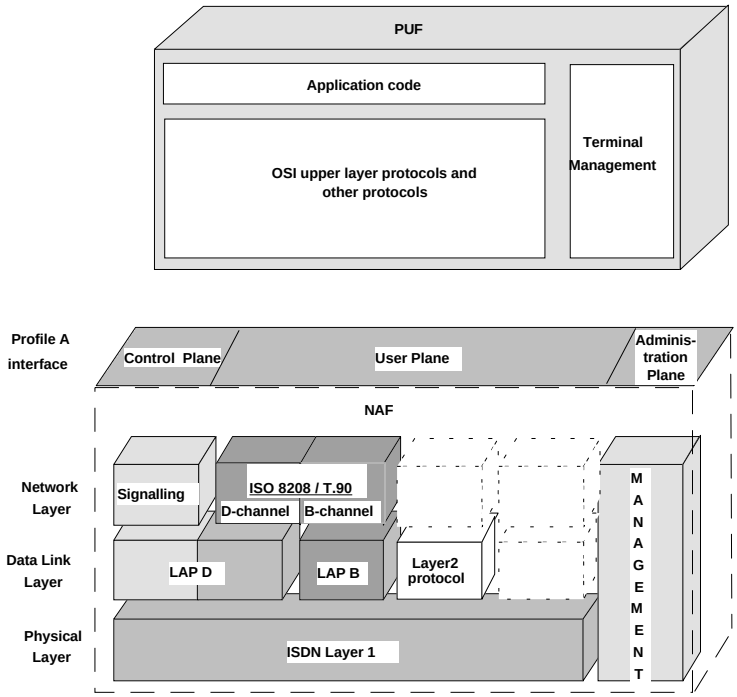


Figure 25: OSI location

General description of the conventions used are provided in subclause 5.2.3.

5.6.4.1.2 Description of messages

The User Plane messages provide an X.213-access to ISO 8208 [3] or ETS 300 080 [1] protocol stacks. The following is a list and short description of significant User Plane messages. Table 43 gives an overview of these messages.

Table 43: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message	used for ISO 8208	used for T.90
301	1	UConnectReq	Requests establishment of a user connection.	X	X
302	1	UConnectInd	Indicates establishment of a user connection has been requested.	X	X
303	1	UConnectRsp	Indicates acceptance of user connection establishment.	X	X
304	1	UConnectCnf	Confirms user connection has been established.	X	X
305	1	UDisconnectReq	Requests removal of user connection.	X	X
306	1	UDisconnectInd	Indicates removal of user connection.	X	X
307	1	UDataReq	Requests data transfer on an established user connection.	X	X
308	1	UDataInd	Indicates arrival of transferred data on an established user connection.	X	X
309	1	UExpeditedDataReq	Requests expedited data transfer on an established user connection.	X	
310	1	UExpeditedDataInd	Indicates presence of transferred expedited data on an established user connection.	X	
311	1	UResetReq	Requests reset to initial state of an established user connection.	X	X
312	1	UResetInd	Indicates reset to initial state of an established user connection.	X	X
313	1	UResetRsp	Indicates acceptance of reset to initial state of an established user connection.	X	X
314	1	UResetCnf	Confirms acceptance of reset to initial state of an established user connection.	X	X
315	1	UDataAcknowledgeReq	Requests acknowledgement of data received on an established user connection.	X	
316	1	UDataAcknowledgeInd	Indicates acknowledgement of data transferred on an established user connection.	X	
317	1	UReadyToReceiveReq	Used to perform flow control for a user connection.	X	X
318	1	UReadyToReceiveInd	Used to indicate flow control status on a user connection.	X	X

5.6.4.1.2.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
CalledDTEAddress	O	If provided, this value supersedes the NCO value.
CalledDTEAddressExt	O	If provided, this value supersedes the NCO value.
CallingDTEAddress	O	If provided, this value supersedes the NCO value.
CallingDTEAddressExt	O	If provided, this value supersedes the NCO value.
ReceiptConfirm	O	Used to request confirmation of data receipt for this user connection.
ExpeditedData	O	Used to request use of expedited data for the user connection.
QOSParameters	O	Quality of Service.
UserData	O	Maximum length is 16, or 128 if FastSelect parameter is used.
Bcug	O	Used to specify the Bilateral Closed User Group facility. If specified then Called address parameters are not allowed.
FastSelect	O	If used this parameter invokes the use of the Fast Select facility.
PacketSize	O	Requested value, overrides any value specified as part of NCO creation.
WindowSize	O	Requested value, overrides any value specified as part of NCO creation.
FacilityData	O	Used to supply facilities. If present, the following facilities shall be overridden by information found elsewhere in this message: - BCUG; - FastSelect; - Called Address Extension; - Calling Address Extension.

Related: UConnectCnf.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
CalledDTEAddress	O	Called address.
CalledDTEAddressExt	O	Called address extension.
CallingDTEAddress	O	Calling address.
CallingDTEAddressExt	O	Calling address extension.
ReceiptConfirm	O	Indicates if confirmation of data receipt is required on this user connection
ExpeditedData	O	Indicates if use of expedited data is allowed on this user connection.
QOSParameters	O	Quality of Service.
UserData	O	Maximum length is 16, or 128 if FastSelect parameter is present.
Bcug	O	Used to pass Bilateral Closed User Group facility information. If present then addressing information shall not be present.
FastSelect	O	Authorisation type to transmit UserData.
PacketSize	M	Value to be used for this user connection.
WindowSize	M	Value to be used for this user connection.
FacilityData	O	Used to supply facilities. The following facilities, if present, are presented by the use of specific parameters: - BCUG; - FastSelect; - Called Address Extension; - Calling Address Extension.

Related: UConnectRsp.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
CalledDTEAddress	O	Called address.
CalledDTEAddressExt	O	Called address extension.
CallingDTEAddress	O	Calling address.
CallingDTEAddressExt	O	Calling address extension.
RespondingDTEAddress	O	The address used to accept the user connection. This may be different from the original called address.
RespondingDTEAddressExt	O	The address extension used to accept the user connection. This may be different from the original called address extension.
ReceiptConfirm	O	Used to accept or not accept use of receipt confirmation for data on this user connection.
ExpeditedData	O	Used to accept or not accept use of expedited data on this user connection.
QOSParameters	O	Quality of Service.
UserData	O	Maximum length is 16, or 128 if FastSelect parameter was present on UConnectInd.
PacketSize	O	Used to indicate agreed value.
WindowSize	O	Used to indicate agreed value.
FacilityData	O	Used to supply facilities.

Related: UConnectInd.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF on the establishment of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
CalledDTEAddress	O	Called address.
CalledDTEAddressExt	O	Called address extension.
CallingDTEAddress	O	Calling address.
CallingDTEAddressExt	O	Calling address extension.
RespondingDTEAddress	O	The address used to accept the user connection. This may be different from the original called address.
RespondingDTEAddressExt	O	The address extension used to accept the user connection. This may be different from the original called address.
ReceiptConfirm	O	Indicates if receipt confirmation of data can be used on this user connection.
ExpeditedData	O	Indicates if expedited data can be used on this user connection.
QOSParameters	O	Quality of Service.
UserData	O	Maximum length is 16, or 128 if FastSelect parameter was present on UConnectReq.
PacketSize	M	Value to be used for this user connection.
WindowSize	M	Value to be used for this user connection.
FacilityData	O	Used to supply facilities.

Related: UConnectReq.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
X213Cause	O	X.213 reason to remove the user connection. Both X213Cause and X25Cause cannot be used on the same message. If neither X213Cause and X25Cause are supplied the X213Cause parameter with the value of disconnection-normal condition shall be used.
RespondingDTEAddress	O	The address used to accept the user connection. This may be different from the original called address.
RespondingDTEAddressExt	O	The address extension used to accept the user connection. This may be different from the original called address.
UserData	O	Only allowed if FastSelect parameter was specified during the user connection establishment. Maximum size of 128 octets.
X25Cause	O	Reason to remove the user connection. Both X213Cause and X25Cause cannot be used on the same message.
X25Diagnostic	C	Complementary information for reason. Optional if X25Cause parameter supplied otherwise not allowed.
FacilityData	O	Used to supply facilities.
NOTE ITU-T Recommendation X.213 [6] cause is exclusive with ITU-T Recommendation X.25 information. If ITU-T Recommendation X.25 cause, optionally associated with the X.25 diagnostic is used, the X.213 cause shall not appear.		

Related: None.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
X213Origin	M	Identifies the initiator of the user connection removal.
X213Cause	O	X.213 Reason to remove the user connection.
UserData	O	Only allowed if FastSelect parameter was specified during the user connection establishment. Maximum size of 128 octets.
RespondingDTEAddress	O	The address used to accept the user connection. This may be different from the original called address extension.
RespondingDTEAddressExt	O	The address extension used to accept the user connection. This may be different from the original called address extension.
X25Cause	O	Reason to remove the user connection. Both X213Cause and X25Cause cannot be used on the same message.
X25Diagnostic	C	Complementary information for Reason. Optional if X25Cause parameter supplied otherwise not allowed.
FacilityData	O	Used to supply facilities.
NOTE: ITU-T Recommendation X.213 [6] cause is exclusive with ITU-T Recommendation X.25 information. If ITU-T Recommendation X.25 cause, optionally associated with the ITU-T Recommendation X.25 diagnostic is used, the ITU-T Recommendation X.213 [6] cause shall not appear.		

Related: None.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
Bit_DQM	O	Used to set the Qualifier bit, the More Bit and to request confirmation of receipt of data.

Remark: Data to send is mandatory. It is not provided as a parameter of the message. Mandatory data shall be provided in the data buffer.

Related: UReadyToReceiveInd.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
Bit_DQM	O	Used to indicate the Qualifier bit value, the More Bit value and the need of confirmation of data reception.

Remark: Data received is always provided, but not as a parameter of the message. Data is provided in the data buffer. This buffer, in this case, is mandatory.

Related: UReadyToReceiveReq.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.9 UExpeditedDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send expedited data. This data is not constrained by the flow control mechanism used to control UDataReq messages.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
UserData	M	Expedited data to transfer.

Related: None.

Protocols: This message is used in the User Plane protocol ISO 8208 [3].

5.6.4.1.2.10 UExpeditedDataInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF the reception of expedited data. This data was not constrained by the flow control mechanisms used to control UDataInd messages.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
UserData	M	Expedited data received.

Related: None.

Protocols: This message is used in the User Plane protocol ISO 8208 [3].

5.6.4.1.2.11 UResetReq

Class: 1 (Basic Class).

Description: This message allows the PUF to reset a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
X213Cause	O	X.213 Reason to reset the user connection. If neither X213Cause and X25Cause are supplied the X213Cause parameter with the value of disconnection-normal condition will be used.
X25Cause	O	Reason to reset the user connection.
X25Diagnostic	C	Complementary information. Optional only if X25Cause supplied, otherwise not allowed.
NOTE: X.213 cause is exclusive with X.25 information. If X.25 cause, optionally associated with the X.25 diagnostic, is used the X.213 cause shall not appear.		

Related: UResetCnf.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.12 UResetInd

Class: 1 (Basic Class).

Description: This message informs the PUF of the reset of a user connection.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
X213Origin	M	Identifies the initiator of the reset user connection.
X213Cause	O	X213 Reason to reset the user connection.
X25Cause	O	Reason to reset the user connection.
X25Diagnostic	C	Complementary information. Optional only if X25Cause supplied, otherwise not allowed.
NOTE: X.213 cause is exclusive with X.25 information. If X.25 cause, optionally associated with the X.25 diagnostic is used, the X.213 cause shall not appear.		

Related: UResetRsp.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.13 UResetRsp

Class: 1 (Basic Class).

Description: This message allows the PUF to respond to a user connection reset, indicating that it has dealt with the reset and is ready to proceed.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UResetInd.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.14 UResetCnf

Class: 1 (Basic Class).

Description: This message completes the reset operation of a user connection. The PUF is now able to transfer data once again.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UResetReq.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.15 UDataAcknowledgeReq

Class: 1 (Basic Class).

Description: This message allows the PUF to acknowledge received data. It should be used when a UDataInd message is received with the Bit_DQM parameter set indicating receipt of confirmation is required.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UDataInd.

Protocols: This message is used in the User Plane protocol ISO 8208.

5.6.4.1.2.16 UDataAcknowledgeInd

Class: 1 (Basic Class).

Description: This message informs the PUF of the reception of an acknowledgement for transferred data. It acknowledges a UDataReq message that was sent with the Bit_DQM parameter requesting confirmation of data reception.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UDataReq.

Protocols: This message is used in the User Plane protocol ISO 8208.

5.6.4.1.2.17 UReadyToReceiveReq

Class: 1 (Basic Class).

Description: This message allows the PUF to indicate to the NAF if it can accept incoming data (UDataInd message). This message can only apply to an already established user connection. Setting the ReadyFlag parameter to TRUE allows the NAF to transfer incoming data to the PUF. Setting the ReadyFlag to FALSE inhibits the transfer.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the PUF is ready to accept incoming data.

Remarks: For a given connection, if more than one message with the same flag value is sent, it shall be ignored by the NAF.

Related: UDataInd.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.2.18 UReadyToReceiveInd

Class: 1 (Basic Class).

Description: This message allows the NAF to indicate to the PUF if the user connection permits the sending of data (UDataReq messages). This message can only apply to an already established user connection. If the ReadyFlag parameter value is FALSE, the NAF can not send data. If the value is TRUE the NAF indicates that data transfer is allowed.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the NAF is ready to receive data for transmission on a user connection.

Related: UDataReq.

Protocols: This message is used in the two User Plane protocols ETS 300 080 [1] and ISO 8208 [3].

5.6.4.1.3 Messages parameters

This subclause describes parameters for the ISO 8208 User Plane and the ETS 300 080 User Plane. They are alphabetically ordered.

Information encoding is provided in subclause 5.2.2.

Table 44: Overview of user parameters

Param. Identif.	Parameter Name	Use in ISO 8208 user messages	Use in ETS 300 080 user messages	Use in UAttributeSet	Other use
1	Algorithm				X
2	Bilateral closed user group	X	X		
4	Bit_DQM	X	X (note)		
5	CalledDTEAddress	X	X		
6	CalledDTEAddressExt	X	X		
9	CallingDTEAddress	X	X		
10	CallingDTEAddressExt	X	X		
29	ExpeditedData	X			
31	FacilityData	X	X (note)		
32	FastSelect	X	X (note)	X	
33	GroupID				X
38	L2ConnectionMode			X	
39	L2FrameSize			X	
40	L2WindowSize			X	
41	L2XID			X	
42	L3ConnectionMode			X	
43	L3IncomingCount			X	
44	L3OutgoingVCCount			X	
45	L3TwoWayCount			X	
50	NCOType				X
52	PacketSize	X	X	X	
54	QOSParameters	X	X	X	
55	ReadyFlag	X	X		
57	ReceiptConfirm	X			
58	RespondingDTEAddress	X	X		
59	RespondingDTEAddressExt	X	X		
61	TEI			X	
62	UProtocol			X	
63	UAttributeName				X
64	UDirection				X
65	UserData	X			
67	WindowSize	X	X	X	
68	X213Cause	X	X		
69	X213Origin	X	X		
70	X25Cause	X	X		
71	X25Diagnostic	X	X		

NOTE: This parameter shall be used in accordance with the rules of ETS 300 080 [1].

5.6.4.1.3.1 Algorithm

Description: This parameter is used to pass the name of the security algorithm to be used to the NAF.

Type: 1.

Fields	Field type	Direction	Required	Comment
Algorithm	IA5-string	P	M	The security algorithm is identified by its name. The names of the available algorithms can be obtained using the Property information. 'nosecurity': this value for this parameter indicates that security is no longer needed for the connection. 16 bytes is the maximum length.

5.6.4.1.3.2 Bilateral closed user group (Bcug)

Description: This parameter is used to pass Bilateral closed user group information to/from the PUF.

Type: 2.

Fields	Field type	Direction	Required	Comment
Bcug	Octet-string	B	M	Index to bilateral closed user group selected for user connection. 4 octets is the fixed length.

5.6.4.1.3.3 Bit_DQM

Description: This parameter is used to pass to/from the PUF:

- need for receipt of data (bit 1). This bit is equivalent to the X.25 D bit;
- Qualifier bit value (bit 2);
- More Data bit value (bit 3).

Each information use a binary position. The Most Significant Bit (MSB) if the bit 8 and the Least Significant Bit is the bit 1. Bit 1 is for value 1, bit 2 for value 2 and bit 3 for value 4. The result value applying to this parameter is the sum of the value for each bit (logical OR).

Type: 4.

Fields	Field type	Direction	Required	Comment
DQM	Octet	B	M	Bit 1: 1 - Confirmation of data reception is allowed or required. 0 - Confirmation of data reception is not allowed or not required. Bit 2: 1 - Set Qualifier bit. 0 - Reset Qualifier bit Bit 3: 1 - Set More bit. 0 - Reset More bit

Remarks: Invalid use of the More bit with the Qualifier bit shall result in the user connection being reset.

For ETS 300 080 protocol, this parameter shall be used in accordance with the rules of ETS 300 080 [1].

5.6.4.1.3.4 CalledDTEAddress

Description: This parameter is used to pass remote DTE address information to/from the PUF.

Type: 5.

Fields	Field type	Direction	Required	Comment
Address	IA5-string	B	M	15 octets is the maximum length

Remark: The BCD translation is provided by the NAF.
In the message exchange from PUF to NAF this parameter shall either be supplied in the NCO or in the appropriate message.

5.6.4.1.3.5 CalledDTEAddressExt

Description: This parameter is used to pass remote DTE address extension information to/from the PUF.

Type: 6.

Fields	Field type	Direction	Required	Comment
AddressExt	IA5-string	B	M	40 octets is the maximum length.

Remark: The BCD translation is provided by the NAF.

5.6.4.1.3.6 CallingDTEAddress

Description: This parameter is used to pass local DTE address information to/from the PUF.

Type: 9.

Fields	Field type	Direction	Required	Comment
Address	IA5-string	B	M	15 octets is the maximum length.

Remark: The BCD translation is provided by the NAF.

5.6.4.1.3.7 CallingDTEAddressExt

Description: This parameter is used to pass local DTE address extension information to/from the PUF.

Type: 10.

Fields	Field type	Direction	Required	Comment
AddressExt	IA5-string	B	M	40 octets is the maximum length

Remark: The BCD translation is provided by the NAF.

5.6.4.1.3.8 ExpeditedData

Description: This parameter is used to pass use of expedited data information to/from the PUF.

Type: 29.

Fields	Field type	Direction	Required	Comment
Usage	Boolean	B	M	TRUE - Use of expedited data is required or supported. FALSE - Use of expedited data is not required or not supported.

5.6.4.1.3.9 FacilityData

Description: This parameter is used to pass facility information to/from the PUF.

Type: 31.

Fields	Field type	Direction	Required	Comment
FacilityData	Octet string	B	M	Encoded as facility information defined in ISO 8208 [3]. 109 octets is the maximum length.

5.6.4.1.3.10 FastSelect

Description: This parameter is used to pass Fast Select Facility information to/from the PUF.

Type: 32.

Fields	Field type	Direction	Required	Comment
FastSelect	Octet	B	M	non-restricted (1) - Called DTE is not required to remove the user connection before establishment is complete. restricted (2) - Called DTE is required to remove the user connection before establishment is complete.

Remarks: When specified on a UConnectReq message this parameter allows the UserData parameter to have a maximum length of 128 octets. If the *restricted* option is selected it indicates that the user connection cannot be established and that a UDisconnectInd should be expected with a maximum UserData parameter of 128 octets. If the *nonrestricted* option is specified then the user connection can be established and the subsequent UConnectCnf can have a maximum UserData parameter of 128 octets. Subsequent to this both the UDisconnectInd and UDisconnectReq fields may also have maximum UserData parameters of 128 octets.

When received on a UConnectInd message this parameter indicates that the UserData parameter within the message can have a maximum length of 128 octets. If the *restricted* option is selected it indicates that the user connection cannot be established and that the PUF shall respond with UDisconnectReq. The UserData parameter with this message can have a maximum length of 128 octets. If the *non-restricted* option is selected, the PUF can respond with UConnectRsp with a maximum UserData parameter of 128 octets Subsequently, both the UDisconnectInd and UDisconnectReq fields may also have maximum UserData parameters of 128 octets.

5.6.4.1.3.11 GroupID

Description: This parameter is used to pass the group identifier to/from the PUF.

Type: 33.

Fields	Field type	Direction	Required	Comment
GroupID	Octet string	B	M	The value is unique for a PUF/NAF relation. 4 octets is the fixed length.

5.6.4.1.3.12 L2ConnectionMode

Description: This parameter is used to pass details of the layer connection mode to the NAF.

Type: 38.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	dte (1) - Act as DTE as defined in ISO 7776 dce (2) - Act as DCE as defined in ISO 7776 auto (3) - When calling act as DTE, when called act as DCE

5.6.4.1.3.13 L2FrameSize

Description: This parameter is used to pass details of the layer 2 frame size to the NAF.

Type: 39.

Fields	Field type	Direction	Required	Comment
Value	Octet string	P	M	Frame size in octets. Length is fixed to 2 octets. The first octet contains the most significant byte of the 2 bytes containing the value.

5.6.4.1.3.14 L2WindowSize

Description: This is used to pass details of the layer 2 window size to the NAF.

Type: 40.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Window size

5.6.4.1.3.15 L2XID

Description: This is used to pass details of the layer 2 XID value and its use.

Type: 41.

Fields	Field type	Direction	Required	Comment
Use	Octet	P	M	send (1) - send XID. match (2) - match XID with XID received. IF XID does not match connection shall not be established.
Value	Octet-string	P	M	XID value [Identifier and signature]. Maximum length is 64 octets.

5.6.4.1.3.16 L3ConnectionMode

Description: This parameter is used to pass details of the layer connection mode to the NAF.

Type: 42.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	dte (1) - act as DTE. dce (2)- act as DCE. auto (3)- act as DTE when calling, act as DCE when called. dxe (4)- use Restart Packet to determine DTE or DCE role as in ISO 8208 auto.

5.6.4.1.3.17 L3IncomingVCCount

Description: This parameter is used to pass the number of connections that may be established at any time by incoming call establishment requests.

Type: 43.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	P	M	Number of connections. Maximum value is 4 095. Length is fixed to 2 octets.

5.6.4.1.3.18 L3OutgoingVCCount

Description: This parameter is used to pass the number of connections that may be established at any instant by outgoing call establishment requests.

Type: 44.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	P	M	Number of connections. Maximum value is 4095. Length is fixed to 2 octets.

5.6.4.1.3.19 L3TwoWayVCCount

Description: This parameter is used to pass the number of connections that may be established at any instant by outgoing or incoming connection establishment requests.

Type: 45.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	P	M	Number of connections. Maximum value is 4095. Length is fixed to 2 octets.

5.6.4.1.3.20 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	U3 (2) - network user access with NAF signalling co-ordination (NAF co-ordination functionality). C/U (3) - signalling and network layer user access. U3G (4) - network user access to additional virtual circuits. This NCO shall be grouped to an existing U3 or C/U type NCO.

5.6.4.1.3.21 PacketSize

Description: This parameter is used to pass packet size information to/from the PUF.

Type: 52.

Fields	Field type	Direction	Required	Comment
Negotiation	Boolean	B	M	Used to indicate if negotiation of packet size is possible. TRUE - negotiation possible. FALSE - negotiation not possible.
Invalue	Octet	B	M	Inbound maximum user data length (see table 44). Maximum size of data that can be received with UDataInd.
Outvalue	Octet	B	M	Outbound maximum user data length (see table 44). Maximum size of data that can be passed with UDataReq.

Remarks: This parameter is used to determine the maximum size of data buffers that can be passed with the UDataReq and UDataInd messages. It is used as follows:

- on UConnectReq the PUF may specify the values it wishes to use;
- on UConnectCnf the NAF shall always specify the values to be used for the user connection;
- on UConnectInd the NAF shall always indicate the values to be used for the user connection. It also indicates if it is possible for the PUF to negotiate these values;
- on UConnectRsp the PUF can specify values if the UConnectInd indicated that negotiation was possible.

For ETS 300 080 protocol, this parameter shall be used in accordance with the rules of ETS 300 080 [1].

Table 45: Precoded packet size values

Precoded value	Packet size (octet)	Precoded value	Packet size (octet)
4	16	9	512
5	32	10	1 024
6	64	11	2 048
7	128	12	4 096
8	256		

5.6.4.1.3.22 QOSParameters

Description: This parameter is used to pass Quality of Service information to/from the PUF.

Type: 54.

Fields		Field type	Direction	Required	Comment
Throughput	Usage	Boolean	B	M	Indicates if following values are included.
	InTarget	Octet	B	C	Values provided in the table 45.
	InLowest	Octet	B	C	Values provided in the table 45.
	InAvailable	Octet	B	C	Values provided in the table 45.
	InSelected	Octet	B	C	Values provided in the table 45.
	OutTarget	Octet	B	C	Values provided in the table 45.
	OutLowest	Octet	B	C	Values provided in the table 45.
	OutAvailable	Octet	B	C	Values provided in the table 45.
	OutSelected	Octet	B	C	Values provided in the table 45.
NCPriority	Usage	Boolean	B	M	Indicates if following values are included.
	Target	Octet	B	C	(note 1).
	Lowest	Octet	B	C	(note 1).
	Available	Octet	B	C	(note 1).
	Selected	Octet	B	C	(note 1).
TransitDelay	Usage	Boolean	B	M	Indicates if following values are included.
	Selected	Octet-string	B	C	(note 2).
	Target	Octet-string	B	C	(note 2).
	Maximum	Octet-string	B	C	(note 2). Conditional if Target - previous one - used, otherwise absent.
End to End Transit Delay	Usage	Boolean	B	M	Indicates if following values are included.
	Selected	Octet-string	B	C	(note 2).
	Target	Octet-string	B	C	(note 2).
	Maximum	Octet-string	B	C	(note 2). Conditional if Target - previous one - used, otherwise absent.

NOTE 1: The NCPriority fields can take any value from 1 (highest priority) to 10 (lowest priority). If not used, the field shall be filled with the value 0. If unspecified, the field shall be filled with the value 11.

NOTE 2: Length is fixed to 2. The lower octet contains least significant byte. 65535 means not used. Delay is expressed in milliseconds.

Remarks: For ETS 300 080 protocol, this parameter shall be used in accordance with the rules of ETS 300 080 [1].

Table 46: Throughput precoding value

Precoding value	Throughput class	Precoding value	Throughput class
3	75	9	4 800
4	150	10	9 600
5	300	11	19 200
6	600	12	48 000
7	1 200	13	64 000
8	2 400	0	unused

5.6.4.1.3.23 ReadyFlag

Description: This parameter is used to request and indicate flow control status on a user connection.

Type: 55.

Fields	Field type	Direction	Required	Comment
Usage	Boolean	B	M	TRUE - Data transfer is allowed. FALSE - Data transfer is not allowed.

5.6.4.1.3.24 ReceiptConfirm

Description: This parameter is used to request confirmation of data receipt for a User Plane connection.

Type: 57.

Fields	Field type	Direction	Required	Comment
Value	Boolean	B	M	TRUE - Confirmation requested FALSE - Confirmation not requested

5.6.4.1.3.25 RespondingDTEAddress

Description: This parameter is used to pass responding DTE address information to/from the PUF.

Type: 58.

Fields	Field type	Direction	Required	Comment
Address	IA5-string	B	M	16 octets is the maximum length.

5.6.4.1.3.26 RespondingDTEAddressExt

Description: This parameter is used to pass responding DTE address extension information from/to the PUF.

Type: 59.

Fields	Field type	Direction	Required	Comment
AddressExt	IA5-string	B	M	40 octets is the maximum length

5.6.4.1.3.27 TEI

Description: This parameter is used to access a permanent link to a data packet switch (packet connection in D-channel).

Type: 61.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	

5.6.4.1.3.28 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	Default (255) - ETS 300 080 [1] ETS 300 080 (1) [1] ISO 8208 (2) [3]
L2Protocol	Octet	P	O	Default (255) - ISO 7776 [4]
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access.

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.4.1.3.29 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.4.1.3.30 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	M	listen (1) call (2) both (3)

5.6.4.1.3.31 UserData

Description: This parameter is used to pass data that is limited in size to/from the PUF.

Type: 65.

Fields	Field type	Direction	Required	Comment
Data	Octet-string	B	M	128 octets is the maximum length. The maximum length allowed varies from message to message and is also different depending on the use of the FastSelect parameter.

5.6.4.1.3.32 WindowSize

Description: This parameter is used to pass window size information to/from the PUF.

Type: 67.

Fields	Field type	Direction	Required	Comment
Negotiation	Boolean	B	M	Used to indicate if negotiation of window size is possible. TRUE - negotiation possible. FALSE - negotiation not possible.
Invalue	Octet	B	M	Inbound window size.
Outvalue	Octet	B	M	Outbound window size.

Remarks: This parameter is used to determine the window sizes to be used for a user connection.

- On UConnectReq, the PUF may specify the values it wishes to use.
- On UConnectCnf, the NAF shall always specify the values to be used for the user connection.
- On UConnectInd the NAF shall always indicate the values to be used for the user connection. It also indicates if it is possible for the PUF to negotiate these values.
- On UConnectRsp the PUF can specify values if the UConnectInd indicated that negotiation was possible.

5.6.4.1.3.33 X213Cause

Description: This parameter is used to pass X.213 Cause information to/from the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See User Plane return code values in subclause 5.6.4.1.7.3.

5.6.4.1.3.34 X213Origin

Description: This parameter is used to pass X.213 origin information to/from the PUF.

Type: 69.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	undefined (1) NAF Provider (2) PUF User (3)

5.6.4.1.3.35 X25Cause

Description: This parameter is used to pass X.25 Cause information to/from the PUF.

Type: 70.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See ISO 8208 [3] cause code values.

5.6.4.1.3.36 X25Diagnostic

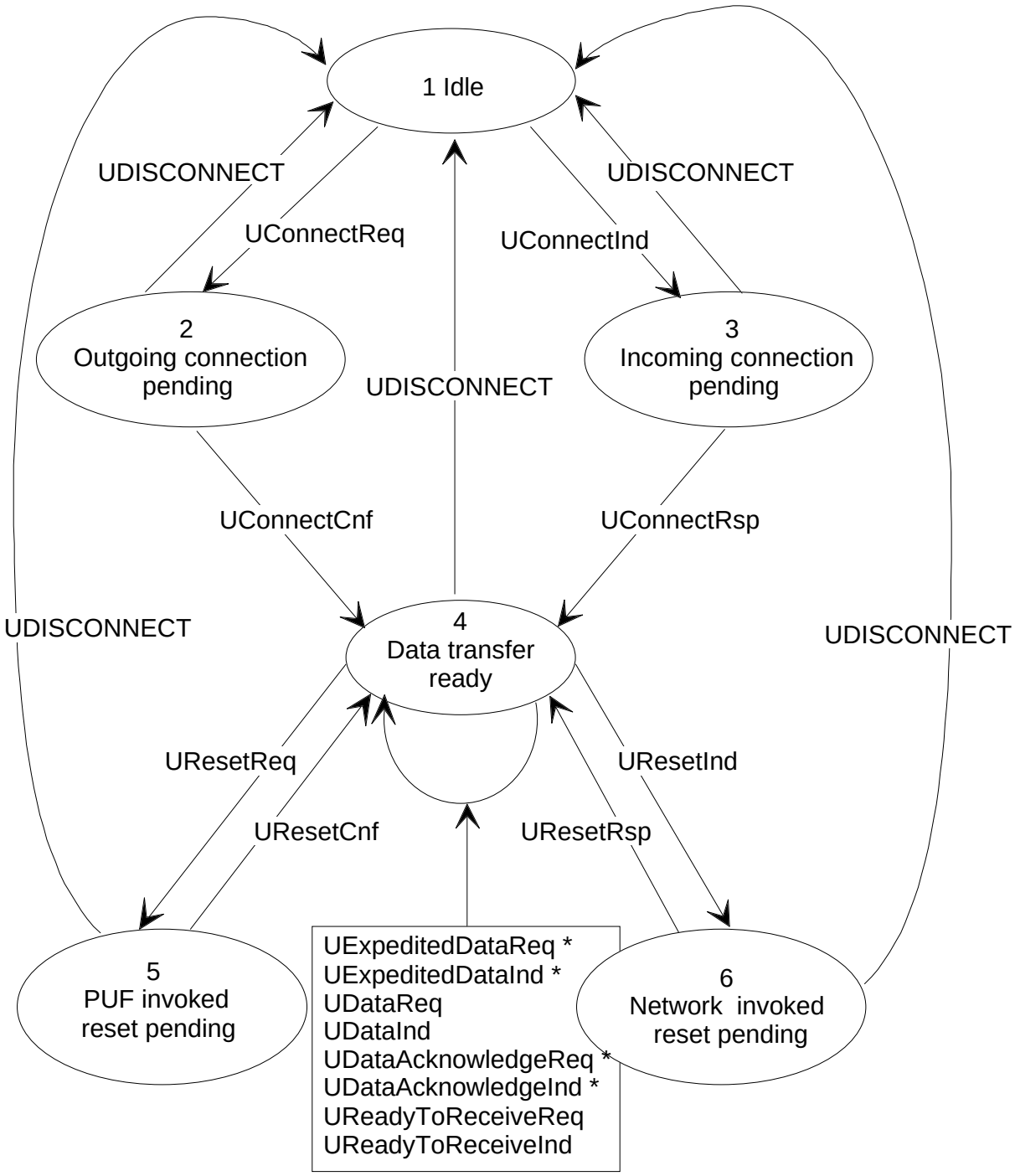
Description: This parameter is used to pass X.25 Diagnostic information to/from the PUF.

Type: 71.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	See ISO 8208 [3] diagnostic values.

5.6.4.1.4 State diagram

Figure 26 shows the different states of a user connection using the user received messages and in which order these messages shall be used.



* this message is used for the ISO 8208 protocol only

NOTE: Where UDISCONNECT appears it can be either UDisconnectReq or UDisconnectInd.

Figure 26: Overview of the User Plane messages

5.6.4.1.5 Co-ordination function

The co-ordination function may be used. See subclause 7.4 for details.

5.6.4.1.6 Selection criteria

This subclause deals with ISO 8208 [3] specific parameters. General NCO criteria are provided in subclause 5.8.

5.6.4.1.6.1 NCO Selection

To select a NCO, the NAF uses the following parameters:

- packet size negotiation;
- window size negotiation.

5.6.4.1.6.1.1 Packet size negotiation

In the INCOMING CALL packet, if the packet size is not provided, the default value, i.e. 128 octets is assumed.

The NCO packet size is correct if one of the following cases is relevant:

- the packet size - provided in the UAttributeSet - is equal to the packet size provided in the INCOMING CALL packet or assumed;

- if there is no packet size provided in the UAttributeSet.

5.6.4.1.6.1.2 Window size negotiation

In the INCOMING CALL packet, if the Window size is not provided, the default value, i.e. 2, is assumed.

The NCO window size is correct if one of the following cases is relevant:

- the window size - provided in the UAttributeSet - is equal to the window size provided in the INCOMING CALL packet or assumed;

- if there is no window size provided in the UAttributeSet.

5.6.4.1.6.1.3 Effective packet size and window size negotiation

In the UConnectRsp, if packet size is not provided, the packet size provided in the incoming call - i.e. UConnectInd - is accepted by the PUF. The same rules apply to the window size.

In the UConnectCnf, if the packet size/window size is not provided, the packet size/window size provided during the outgoing call - i.e. UConnectReq - is approved for use by the PUF.

5.6.4.1.6.2 Action if no NCO available

A disconnect with the X213Reason "Connection rejection - reason unspecified transient" is issued by the NAF.

5.6.4.1.7 Specific error handling and codes

Errors are dealt as given in subclauses 5.6.4.1.7.1 to 5.6.4.1.7.3.

5.6.4.1.7.1 Invalid use of User Plane messages

In case of:

- invalid use of Receipt Confirmation Service;
- invalid use of Confirmation request on UDataReq;
- invalid length of UDataReq UserData parameter;
- invalid use of Expedited Data;
- invalid issuing of messages while in Reset state.

action is:

- PUF is sent UDisconnectInd.

In case of:

- invalid Use of Bit_DQM (association between More and Qualifier bits) parameters on subsequent UDataReq messages.

action is:

- PUF is sent UResetInd.

5.6.4.1.7.2 Other errors

In the case of parameter content error, PUF is sent UDisconnectInd.

5.6.4.1.7.3 Causes

These values can be specified and are returned in the X213Cause parameter.

Table 47: X213Cause parameter value

Return Code		Meaning	ErrorSpecific Information
Undefined	220	Undefined error situation.	Not present
NSAPunreachablePerm	221	Connection Rejection - NSAP unreachable/fixed condition.	Not present
DiscTrans	225	Disconnection - transient condition.	Not present
DiscPerm	226	Disconnection - fixed condition.	Not present
NoReasonTrans	227	Connection Rejection - reason unspecified/transient condition.	Not present
NoReasonPerm	228	Connection Rejection - reason unspecified/fixed condition.	Not present
QOSnotavailTrans	229	Connection Rejection - QOS not available/transient condition.	Not present
QOSnotavailPerm	230	Connection Rejection - QOS not available/fixed condition.	Not present
NSAPunreachableTrans	231	Connection Rejection - NSAP unreachable/transient condition.	Not present
NSAPunknown	232	Connection Rejection - NSAP address unknown (fixed condition).	Not present
DiscNorm	241	Disconnection - normal condition.	Not present
DiscAbnorm	242	Disconnection - abnormal condition.	Not present
ConRejectTrans	244	Connection rejection - transient condition.	Not present
ConRejectPerm	245	Connection rejection - fixed condition.	Not present
ConRejectUserData	248	Connection rejection - incompatible information in UserData parameter.	Not present

5.6.4.1.8 AttributeSet

5.6.4.1.8.1 AttributeSet parameters

Table 48: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
WindowSize	O	Layer 3 window size. See subclause 10.1.3.32.
PacketSize	O	Layer 3 packet size. See subclause 10.1.3.21.
FastSelect	O	Fast select facility. See subclause 10.1.3.10.
QOSParameters	O	Quality of service. See subclause 10.1.3.22.
UProtocol	O	See remark. See also subclause 10.1.3.28.
L3ConnectionMode	O	See remark. See also subclause 10.1.3.16.
L3TwoWayVCCount	O	See remark. See also subclause 10.1.3.19.
L3IncomingVCCount	O	See remark. See also subclause 10.1.3.17.
L3OutgoingVCCount	O	See remark. See also subclause 10.1.3.18.
TEI	O	See remark. See also subclause 10.1.3.27.
L2ConnectionMode	O	See remark. See also subclause 10.1.3.12.
L2WindowSize	O	See remark. See also subclause 10.1.3.14.
L2FrameSize	O	See remark. See also subclause 10.1.3.13.
L2XID	O	See remark. See also subclause 10.1.3.15.

Remark: These parameters can only be used during NCO creation which contains Control Plane information. NCOs that are to be associated with the use of a GroupID may not specify these parameters. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

If parameters are omitted defaults shall be used. The default values are User Plane protocol dependent. The NAF shall supply the correct value depending on the protocol. Default values are described in annex E.

Table 49: User Plane Address Set (UAddressSet) parameters

Parameters	Required	Comment
CalledDTEAddress	O	See subclause 5.6.4.1.3.4 for parameter definition.
CalledDTEAddressExt	O	See subclause 5.6.4.1.3.5 for parameter definition.
CallingDTEAddress	O	See subclause 5.6.4.1.3.6 for parameter definition.
CallingDTEAddressExt	O	See subclause 5.6.4.1.3.7 for parameter definition.

5.6.4.1.8.2 Static attribute content

The attribute sets described below use following conventions:

- Name shall be used with the ACreateNCOReq message;
- all numerical values are decimal values.

Name	: U_ISO8208
WindowSize	: 2
PacketSize	: 128 (byte)
UProtocol	: ISO 8208
L3ConnectionMode	: DXE
L3TwoWayVCCount	: local arrangement
L3IncomingVCCount	: 1
L3OutgoingVCCount	: 1
L2ConnectionMode	: Auto
L2WindowSize	: 7
L2FrameSize	: 128 (byte)
L2XID	: none

Name	: U_TELEMATIC_TERM
WindowSize	: 2
PacketSize	: 128 (byte)
UProtocol	: ETS 300 080
L3ConnectionMode	: DXE
L3TwoWayVCCount	: local arrangement
L3IncomingVCCount	: 0
L3OutgoingVCCount	: 0
L2ConnectionMode	: Auto
L2WindowSize	: 7
L2FrameSize	: 128 (byte)

5.6.4.2 T.70NL protocol

5.6.4.2.1 Introduction

This subclause deals with the T.70 protocol. In this ETS, whenever ITU-T Recommendation T.70 [15] is referenced, T.70NL is implied (Network Layer).

The OSI location of the T.70 protocol is shown in figure 27.

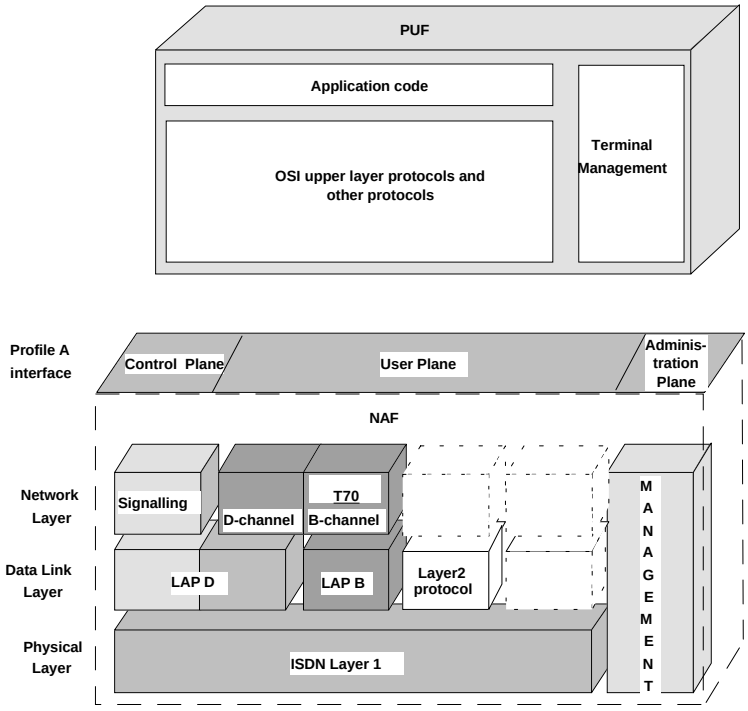


Figure 27: OSI location

General description conventions are provided in subclause 5.2.3.

5.6.4.2.2 Messages

The User Plane messages provide an access to T.70 protocol stacks. The following are a list and short description of relevant User Plane messages. Table 50 gives an overview of these messages.

Table 50: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.

5.6.4.2.2.1 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
Bit_DQM	O	Used to set the ITU-T Recommendation T.70 [15] More Bit and Qualifier bit.

Remark: Data to send are mandatory. They are not provided as a parameter of the message.

Mandatory data shall be provided in the data buffer.

Related: UReadyToReceiveInd.

5.6.4.2.2.2 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
Bit_DQM	O	Used to indicate the ITU-T Recommendation T.70 [15] More Bit and Qualifier bit reception.

Remark: Data received are always provided, but not as a parameter of the message.

Data are provided in the data buffer. This buffer, in this case, is mandatory.

Related: UReadyToReceiveReq.

5.6.4.2.3 Messages parameters

This subclause describes parameters for the T.70 User Plane. They are ordered by parameter identifiers. Information encoding is provided in subclause 5.2.2

Table 51: Overview of user parameters

Param. Identif.	Parameter Name	Use in user messages	Use in UAttributeSet	Other use
4	Bit_DQM	X		
50	NCOType			X
52	PacketSize		X	
62	UProtocol		X	
63	UAttributeName			
64	UDirection			X

5.6.4.2.3.1 Bit_DQM

Description: This parameter is used to pass to/from the PUF:

- More data bit value (bit 3).

  Each information element uses a binary position. The MSB is bit 8 and the Least Significant Bit (LSB) is bit 1. Bit 1 is for value 1, bit 2 for value 2 and bit 3 for value 4. The resulting value applying to this parameter is the sum of the value for each bit (logical OR).

Type: 4.

Fields	Field type	Direction	Required	Comment
DQM	Octet	B	M	Bit 1: 0 - Confirmation of data reception is not allowed or not required. Bit 2: 0 - Reset Qualifier bit Bit 3: 1 - Set More bit. 0 - Reset More bit

5.6.4.2.3.2 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	U3 (2) - network user access with NAF signalling co-ordination (NAF co-ordination functionality).

5.6.4.2.3.3 PacketSize

Description: This parameter is used to pass packet size information to/from the PUF.

Type: 52.

Fields	Field type	Direction	Required	Comment
Negotiation	Boolean	B	M	Used to indicate if negotiation of packet size is possible. TRUE - negotiation possible. FALSE - negotiation not possible.
Invalue	Octet	B	M	Inbound maximum user data length (see table 51). Maximum size of data that can be received with UDataInd.
Outvalue	Octet	B	M	Outbound maximum user data length (see table 51). Maximum size of data that can be passed with UDataReq.

Remarks: This parameter is used to determine the maximum size of data buffers that can be passed using the UDataReq and UDataInd messages.

Table 52: Precoded packet size values

Precoded value	Packet size (octet)	Precoded value	Packet size (octet)
4	16	8	256
5	32	9	512
6	64	10	1 024
7	128	11	2 048

5.6.4.2.3.4 UProtocol

Description: This is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	T.70 (3)
L2Protocol	Octet	P	O	Default (255) - ISO 7776 [4]
L1Protocol	Octet	P	O	Default (255) - Transparent B-channel access

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.4.2.3.5 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.6.4.2.3.6 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.4.2.4 State diagram

User messages do not change the state of the connection.

5.6.4.2.5 Co-ordination function

The co-ordination function can not be used.

5.6.4.2.6 Selection criteria

No T.70 specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.4.2.7 Specific error handling and codes

Protocol errors are not available at the interface.

5.6.4.2.8 Static attributes

5.6.4.2.8.1 AttributeSet parameters

Table 53: User Plane Attribute Set (UAttributeSet) Parameters

Parameters	Required	Comment
UProtocol	O	See remark. See also subclause 5.6.4.2.3.4.
PacketSize	O	See remark. See also subclause 5.6.4.2.3.3.

Remark: These parameters can only be used during NCO creation which contains Control Plane information. Refer to subclause 5.4.1(ACreateNCO operation) for details.

If parameters are omitted defaults shall be used by the NAF. Default values are described in annex E.

5.6.4.2.8.2 Static attribute content

Name	: U_T70
UProtocol	: T.70
PacketSize	: 128

5.6.5 V.120 Protocol

5.6.5.1 Introduction

This clause deals with the V.120 protocol. figure 28 shows the localisation of the user protocol access.

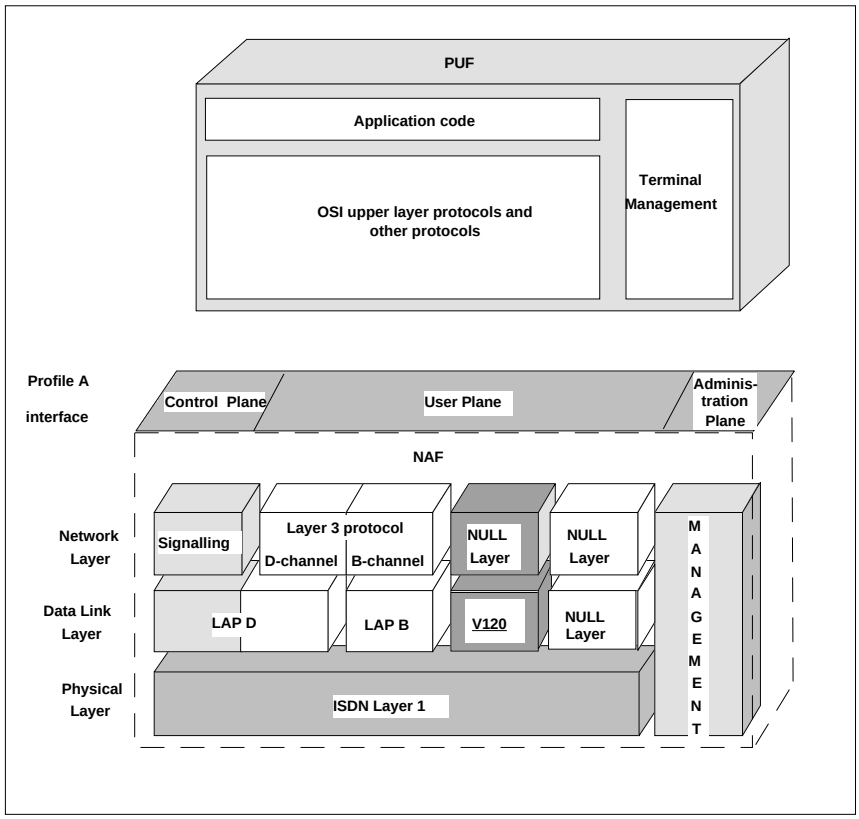


Figure 28: OSI location

General description conventions are provided in subclause 5.2.3.

5.6.5.2 Messages

Table 54 gives an overview of user messages.

Table 54: Overview of user messages

Mess. Identif.	Class	Message Name	Purpose of Message
307	1	UDataReq	Request transfer of data.
308	1	UDataInd	Indicate arrival of transferred data.
315	1	UDataAcknowledgeReq	Request acknowledgement of data received on an established user connection.
316	1	UDataAcknowledgeInd	Indicate acknowledgement of data transferred on an established user connection.
319	1	UErrorInd	Indicate an error.

5.6.5.2.1 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send either the beginning, the end, a part or a complete frame. The Lower Layer Reference is related to the reference negotiated during the channel establishment, if any.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
LLR	O	Lower Layer Reference. Only the first 13 bits will be used.
BlockType	O	Indicates if the block is a type of begin, end, part or complete block. By default, the type is complete.

Remark: Data to send are mandatory. They are not provided as a parameter of the message. Mandatory data shall be provided in the data buffer and the data size is limited to the value which apply to the N201 ETS 300 102 protocol parameter which is generally equal to 260 octets.

Related: UReadyToReceiveInd.

5.6.5.2.2 UDataInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF the reception of the beginning, the end, a part or a complete frame. The Lower Layer Reference is related to the reference negotiated during the channel establishment, if any.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
LLR	O	Lower Layer Reference. Only the first 13 bits will be used.
BlockType	O	Indicates if the block is a type of begin, end, part or complete block. By default, the type is complete.

Remark: Data received are always provided, but not as a parameter of the message. Data are provided in the data buffer. This buffer, in this case, is mandatory and the data size is limited to the value which apply to the N201 ETS 300 102 protocol parameter which is generally equal to 260 octets.

Related: UReadyToReceiveReq.

5.6.5.2.3 UReadyToReceiveReq

Class: 1 (Basic Class).

Description: This message allows the PUF to indicate to the NAF if it can accept incoming data (UDataInd message). This message can only apply to an already established user connection. Setting the ReadyFlag parameter to TRUE allows the NAF to transfer incoming data to the PUF. Setting the ReadyFlag to FALSE inhibits the transfer.

This flow control mechanism does not imply an end to end flow control.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the PUF is ready to accept incoming data.

Remarks: For a given connection, if more than one message with the same flag value is sent, it shall be ignored by the NAF.

Related: UDataInd.

5.6.5.2.4 UReadyToReceiveInd

Class: 1 (Basic class).

Description: This message allows the NAF to indicate to the PUF if the user connection permits the sending of data (UDataReq messages). This message can only apply to an already established user connection. If the ReadyFlag parameter value is FALSE, the NAF can not send data. If the value is TRUE the NAF indicates that data transfer is allowed.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the NAF is ready to receive data for transmission on a user connection.

Related: UDataReq.

5.6.5.2.5 UErrorInd

Class: 1 (Basic Class).

Description: This message indicates to a PUF that an error has occurred. It may be a parity and/or a stop bit error, a FCS error or sequence error.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the Control Plane connection.
Cause	M	Identifies type of error.

Related: None.

5.6.5.3 Messages parameters

This subclause describes parameters for the V.120 plane. Table 55 summarises used parameters.

Table 55: Overview of user parameters

Param. Identif.	Parameter Name	Used in user messages	Used in UAttributeSet	Other use
50	NCOType			X
55	ReadyFlag	X		
62	UProtocol		X	
63	UAttributeName			X
64	UDirection			X
68	Cause	X		
87	LowerLayerReference	X		X
88	BlockType	X		
89	V120FunctionMode		X	

5.6.5.3.1 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	B	M	C/U (3) - signalling and link layer user access.

5.6.5.3.2 ReadyFlag

Description: Flag byte to be sent by the NAF when the user access is idle.

Type: 55.

Fields	Field type	Direction	Required	Comment
Usage	Boolean	B	M	TRUE - Data transfer is possible. FALSE - Data transfer is not possible.

5.6.5.3.3 UProtocol

Description: This is used to select the User Plane protocol. The first octet contains the layer 3 protocol requested, the second octet contains the layer 2 protocol requested and the third octet contains the layer 1 protocol requested.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	NULL (4).
L2Protocol	Octet	P	M	V.120 Recommendation (9).
L1Protocol	Octet	P	M	Default (255) - Transparent B-channel access.

Remark: Other possible values (for other protocols) are provided in subclause 5.6.

5.6.5.3.4 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 is maximum length.

5.6.5.3.5 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	O	both (3)

5.6.5.3.6 Cause

Description: This parameter is used to pass Cause information for disconnection to the PUF.

Type: 68.

Fields	Field type	Direction	Required	Comment
Value	Octet	N	M	213

5.6.5.3.7 LowerLayerReference

Description: This parameter is used to pass V.120 address to/from the PUF. It refers to the Lower Layer Identifier and may by 6 or 13 bits long.

Type: 87.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	B	M	Lower Layer Reference parameter. Length is fixed to 2.

5.6.5.3.8 BlockType

Description: This parameter is used to pass the type of a data block to/from the PUF, in case of V.120 protocol.

Type: 88.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	Type of the block. default (255): complete (1) beginning (2) ending (3) part of a sequence of data block (4) complete data block.

5.6.5.3.9 V120FunctionMode

Description: This parameter is used to pass the way the V.120 protocol will be used.

Type: 89.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	Type of the block. default (255): synchronous (1) asynchronous (2) synchronous (in HDLC) (3) bits transparency.

5.6.5.4 State diagram

User messages do not change the state of the connection.

5.6.5.5 Co-ordination function

The co-ordination function cannot be used with the User Plane protocol relating to V.120 access.

5.6.5.6 Selection criteria

No specific parameters are used. General NCO criteria are provided in subclause 5.8.

5.6.5.7 Specific error handling

Errors are dealt with in the following manner: in case of any error, PUF is sent UErrorInd.

5.6.5.8 Static attributes

5.6.5.8.1 AttributeSet parameters

Table 56: User Plane Attribute Set (UAttributeSet) parameters

Parameters	Required	Comment
V120FunctionMode	O	See subclause 5.6.5.3.9.
UProtocol	O	See subclause 5.6.5.3.3.

Remark: These parameters can only be used during NCO creation containing Control Plane information. Refer to subclause 5.4.1 (ACreateNCO operation) for details.

5.6.5.8.2 Static attribute content

Name	: U_V120
UProtocol	: V120
V120FunctionMode	: synchronous

5.6.5.9 Protocol specific NAF property information

The V.120 specific parameters of NAF-Property are shown in table 57.

Table 57: TLV coded NAF-Property parameter

Parameter	Provided	TLV Coding			Comment and values
		TypeID	Length	Value	
V120FunctionMode	O	16	1	Octet	Indicates the way the V.120 protocol will be used. See subclause 5.6.5.3.9.

See also the PciGetProperty function in subclause 5.3.1.3.

5.6.5.10 Impact on the Control Plane

The Logical Link identifier parameter may be added in the connection messages:

- CConnectReq;
- CConnectInd;
- CConnectCnf;
- CConnectRsp.

Supplementary parameter, in case of V.120:

Name	Required	Comment
Lower Layer Reference	C	Optional if the V120 protocol is used, else absent.

5.6.6 T.30 protocol

This subclause describes how to access the T.30 protocol via Profile A. The T.30 protocol is an User Plane protocol with specific messages and parameters.

5.6.6.1 Overview of T 30 messages

The User Plane messages provide an X.213-access to different protocol stacks. The following is a list and short description of all User Plane messages. Table 58 gives an overview of NMA messages.

Table 58: Overview of NMA messages

Mess. Identif.	Class	Message Name	Purpose of Message
301	1	UConnectReq	Request establishment of a user connection.
302	1	UConnectInd	Indicate establishment of a user connection has been requested.
303	1	UConnectRsp	Indicate acceptance of user connection establishment.
304	1	UConnectCnf	Confirm user connection has been established.
305	1	UDisconnectReq	Request removal of user connection.
306	1	UDisconnectInd	Indicate removal of user connection.
307	1	UDataReq	Request data transfer on an established user connection.
308	1	UDataInd	Indicate arrival of transferred data on an established user connection.
309	1	UExpeditedDataReq	Request expedited data transfer on an established user connection.
310	1	UExpeditedDataInd	Indicate presence of transferred expedited data on an established user connection.
311	1	UResetReq	Request reset to initial state of an established user connection.
312	1	UResetInd	Indicate reset to initial state of an established user connection.
313	1	UResetRsp	Indicate acceptance of reset to initial state of an established user connection.
314	1	UResetCnf	Confirm acceptance of reset to initial state of an established user connection.
315	1	UDataAcknowledgeReq	Request acknowledgement of data received on an established user connection.
316	1	UDataAcknowledgeInd	Indicate acknowledgement of data transferred on an established user connection.
317	1	UReadyToReceiveReq	Used to perform flow control for a user connection.
318	1	UReadyToReceiveInd	Used to indicate flow control status on a user connection.
319	1	UErrorInd	Indicate an error.
320	2	UInformationInd	Used to inform the PUF about the state of the sending or receiving facsimile
321	2	URegisterMailBoxReq	Used to declare a mail box to the NAF
322	2	URegisterMailBoxCnf	Confirm the registration of the mail box to the PUF
323	2	UDestroyMailBoxReq	Used to destroy a mail box previously registered
324	2	UDestroyMailBoxCnf	Confirm whether a mail box was destroyed
326	2	ULocalPollingInd	Used to inform the PUF the remote station has requested to collect a document
327	2	ULocalPollingRsp	Informs the NAF whether the collection of the documents, requested by the remote station, is authorised
329	2	URemotePollingReq	Used to inform the PUF whether the remote station has documents to collect
330	2	URemotePollingInd	Used to request polling on the remote station
331	2	USwitchToVoiceModeReq	Used to inform the NAF that the user has requested a voice contact.
332	2	USwitchToVoiceModeInd	Used to inform the PUF that the remote station made a voice contact request
333	2	USwitchToVoiceModeRsp	Used to inform the NAF that the user accepts switching to speech.
334	2	USwitchToVoiceModeCnf	Used to inform the PUF that the remote operator accepts switching to speech.

Not all of the above messages are used in every cases. Table 59 identifies the messages used in the specific User Plane protocols. A cross placed in the protocol column indicates application of the relevant message to the protocol.

Table 59 gives an overview of NMA messages including the protocol dependencies.

Table 59: Overview of NMA messages (protocol dependencies)

Mess.Identif.	Message Name	ETS 300 080 [1]	ISO 8208 [3]	ITU-T Rec. T.30 [14]
301	UConnectReq	X	X	X
302	UConnectInd	X	X	X
303	UConnectRsp	X	X	X
304	UConnectCnf	X	X	X
305	UDisconnectReq	X	X	X
306	UDisconnectInd	X	X	X
307	UDataReq	X	X	X
308	UDataInd	X	X	X
309	UExpeditedDataReq		X	
310	UExpeditedDataInd		X	
311	UResetReq	X	X	
312	UResetInd	X	X	
313	UResetRsp	X	X	
314	UResetCnf	X	X	
315	UDataAcknowledgeReq		X	X
316	UDataAcknowledgeInd		X	X
317	UReadyToReceiveReq	X	X	X
318	UReadyToReceiveInd	X	X	X
319	UErrorInd			X
320	UInformationInd			X
321	URegisterMailBoxReq			X
322	URegisterMailBoxCnf			X
323	UDestroyMailBoxReq			X
324	UDestroyMailBoxCnf			X
326	ULocalPollingInd			X
327	ULocalPollingRsp			X
329	URemotePollingReq			X
330	URemotePollingInd			X
331	USwitchToVoiceModeReq			X
332	USwitchToVoiceModeInd			X
333	USwitchToVoiceModeRsp			X
334	USwitchToVoiceModeCnf			X

5.6.6.2 Sequencing of User Plane messages

Figure 29 shows the different states a user connection can get using the NMA messages and in which order these messages shall be used.

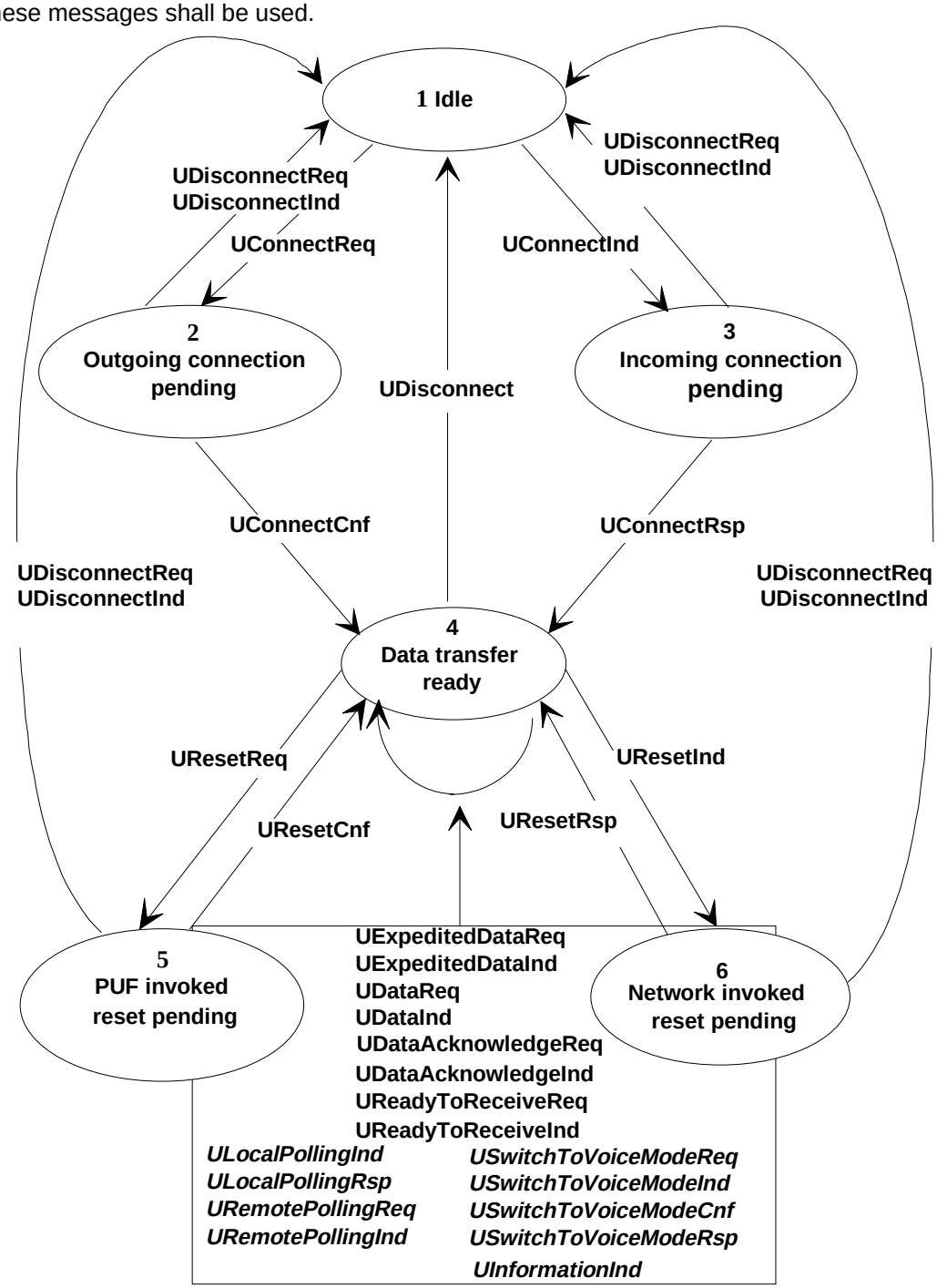


Figure 29: Overview of the User Plane messages

5.6.6.3 Detail of T.30 protocol messages

5.6.6.3.1 UConnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to initiate the establishment of a user connection. It is used to request the establishment of the T.30 protocol over a channel already established.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
CalledDTEAddress	C	This parameter is mandatory. If provided, this value supersedes the NCO value. If absent, it should be included at the NCO creation time.

Related: UConnectCnf.

5.6.6.3.2 UConnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF of an incoming demand to establish a T.30 user connection. It does not ensure of a successful end-to-end negotiation between terminals.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectRsp.

5.6.6.3.3 UConnectRsp

Class: 1 (Basic Class).

Description: This message allows a PUF to accept the establishment of a T.30 user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: UConnectInd.

5.6.6.3.4 UConnectCnf

Class: 1 (Basic Class).

Description: This message informs the PUF on the establishment of a user connection. It does not ensure of a successful end-to-end negotiation between terminals.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.

Related: UConnectReq.

5.6.6.3.5 UDisconnectReq

Class: 1 (Basic Class).

Description: This message allows a PUF to remove a user connection.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
T30Cause	O	T30 reason to remove the user connection.
NOTE In case of the T.30 Protocol this message requests the immediate termination of the protocol.		

Related: None.

5.6.6.3.6 UDisconnectInd

Class: 1 (Basic Class).

Description: This message informs a PUF that a user connection has been removed.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
X213Origin	M	Identifies the initiator of the user connection removal.
T30Cause	O	T30 reason to remove the user connection.

Related: None.

5.6.6.3.7 UDataReq

Class: 1 (Basic Class).

Description: This message allows a PUF to send a data packet. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment. In case of the T.30 Protocol this message allows a PUF to send either data block or data description.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
DataDescription	*O*	*Indicates the characteristics of the page to be transmitted.*
DataBlock	*O*	*Used to pass the transmission data.*

Remark: Data to send are mandatory, except in case of T.30 protocol and if the DataDescription parameter is provided. They are not provided as a parameter of the message.

Related: UReadyToReceiveInd.

5.6.6.3.8 UDataInd

Class: 1 (Basic Class).

Description: This message indicates the presence of received data to a PUF. The size of a data packet is restricted to the data packet size negotiated during the user connection establishment.
In case of T.30 Protocol this message allows a NAF to indicate to a PUF either data block or data description.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
DataDescription	O	Indicates the characteristics of the received page.
DataBlock	O	Used to pass the received data.

Remark: Data received are always provided, but not as a parameter of the message. Data are provided in the data buffer. This buffer, in this case, is mandatory except in case of T.30 protocol and if the DataDescription parameter is provided.

Related: UReadyToReceiveReq.

5.6.6.3.9 UDataAcknowledgeReq

Class: 1 (Basic Class).

Description: This message informs the PUF of the reception of an acknowledgement for transferred data. It acknowledges a page locally received.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
PageAcknowledgement	O	Indicates to the NAF for the complete reception of the current page.

Related: UDataReq.

5.6.6.3.10 UDataAcknowledgeInd

Class: 1 (Basic Class).

Description: This message informs the PUF of the reception of an acknowledgement for transferred data. It indicates the acknowledgement for a page received by the remote part.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
PageAcknowledgement	O	Indicates the PUF whether the remote station has received the current page.

Related: UDataReq.

5.6.6.3.11 UReadyToReceiveReq

Class: 1 (Basic Class).

Description: This message allows the PUF to indicate to the NAF if it can accept incoming data (UDataInd message). This message can only apply to an already established user connection. Setting the ReadyFlag parameter to TRUE allows the NAF to transfer incoming data to the PUF. Setting the ReadyFlag to FALSE inhibits the transfer.

This flow control mechanism does not imply an end to end flow control.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the PUF is ready to accept incoming data.

Remarks: For a given connection, if more than one message with the same flag value is sent, it shall be ignored by the NAF.

Related: UDataInd.

5.6.6.3.12 UReadyToReceiveInd

Class: 1 (Basic Class).

Description: This message allows the NAF to indicate to the PUF if the user connection permits the sending of data (UDataReq messages). This message can only apply to an already established user connection. If the ReadyFlag parameter value is FALSE, the NAF can not send data. If the value is TRUE the NAF indicates that data transfer is allowed.

This flow control mechanism does not imply an end-to-end flow control.

Parameters:

Name	Provided	Comment
NCOID	M	Identifies the user connection.
ReadyFlag	M	This flag indicates whether or not the NAF is ready to receive data for transmission on a user connection.

Related: UDataReq.

5.6.6.3.13 UInformationInd

Class: 2 (Additional Class).

Description: This message informs the PUF about the transmission for received or sent facsimile.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
RemoteDesignation	C (note)	Identifier of the remote station and contents of the non standard fields.
NegociatedCharacteristic	C (note)	Specifies the characteristics of the transmission.
ReceivePageQuality	C (note)	Indicates whether the current page was correctly received.
NOTE: Only one of these parameters should be provided at the same time.		

Related: None.

5.6.6.3.14 URegisterMailBoxReq

Class: 2 (Additional Class).

Description: Request message for creating a mail box. Before using this message the PUF should obtain the NAF-Property to ensure of mail box management.

This message is not related with a established connection and has only a local significance.

Parameters:

Name	Required	Comment
NCOID	M	Identifies a T.30 NCO.
MailBoxType	M	Indicates the type of the mail box.
Password	C	Indicates a password for the polling mode or the use of a subaddress. This parameter is used only if the mail box is used with polling.
PollingNumber	C	Indicates the number of pollings authorised or a coding subaddress. This parameter is used either if mail box is used with polling or if a subaddress is received.
MailBoxMnemonic	O	This parameter is mandatory if the mail box is used when receiving. It can be omitted if the mail box is used when polling.

Related: URegisterMailBoxCnf.

5.6.6.3.15 URegisterMailBoxCnf

Class: 2 (Additional Class).

Description: Confirmation message of RegisterMailBox operation requested previously.

This message may contain the Mail Box identifier which shall be used on further requests.

This message is not related with an established connection and has only a local significance.

Parameters:

Name	Required	Comment
NCOID	M	Identifies a T30 NCO.
CompletionStatus	M	Completion status of the RegisterMailBox operation of the NAF.
MailBoxNumber	C	Mail box identifier if CompletionStatus is success else absent.

Related: URegisterMailBoxReq.

5.6.6.3.16 UDestroyMailBoxReq

Class: 2 (Additional Class).

Description: Destroys an existing mail box create by the same PUF.

This message is not related with an established connection and has only a local significance.

Parameters:

Name	Required	Comment
NCOID	M	Identifies a T30 NCO.
MailBoxNumber	M	Identify the mail box on which the Destroy operation was requested.

Related: UDestroyMailBoxCnf.

5.6.6.3.17 UDestroyMailBoxCnf

Class: 2 (Additional Class).

Description: Confirmation message of the DestroyMailBox operation previously requested.

This message is not related with an established connection and has only a local significance.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
CompletionStatus	M	Completion status of the DestroyMailBox operation of the NAF

Related: UDestroyMailBoxReq.

5.6.6.3.18 ULocalPollingInd

Class: 2 (Additional Class).

Description: This message informs the PUF the remote station has requested to collect a document.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
MailBoxNumber	O	This parameter is used only if the polling indication concerned a specific mail box.

Related: ULocalPollingRsp.

5.6.6.3.19 ULocalPollingRsp

Class: 2 (Additional Class).

Description: This message informs the NAF whether the PUF authorises the collection of the documents requested by the remote station.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
MailBoxNumber	O	Present if provide in LocalPolling indication message.
PollingFlag	M	This flag indicates whether collection is accepted or not.

Related: ULocalPollingInd.

5.6.6.3.20 URemotePollingReq

Class: 2 (Additional Class).

Description: Request message to poll remote station.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: URemotePollingInd.

5.6.6.3.21 URemotePollingInd

Class: 2 (Additional Class).

Description: This message informs the PUF whether the remote station has documents to collect.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
PollingFlag	M	This flag indicates whether the remote station has or not documents to collect (or it refuses to collect them).

Related: None.

5.6.6.3.22 USwitchToVoiceModeReq

Class: 2 (Additional Class).

Description: Request message to switch in voice mode.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: USwitchToVoiceModeCnf.

5.6.6.3.23 USwitchToVoiceModeCnf

Class: 2 (Additional Class).

Description: This message informs the PUF whether the remote station accepts switching to speech.

If accepted, the connection is put in voice mode. The only way to restart the T.30 protocol is to send again an UConnectReq message associated to this NCO.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
SwitchFlag	M	This flag indicates whether the remote accepts switching.

Related: USwitchToVoiceModeReq.

5.6.6.3.24 USwitchToVoiceModeInd

Class: 2 (Additional Class).

Description: This message informs the PUF the remote station has requested to switch in voice mode.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.

Related: USwitchToVoiceModeRsp.

5.6.6.3.25 USwitchToVoiceModeRsp

Class: 2 (Additional Class).

Description: This message informs the NAF whether switching is authorised.

If accepted, the connection is put in voice mode. The only way to restart the T.30 protocol is to send again an UConnectReq message associated to this NCO.

Parameters:

Name	Required	Comment
NCOID	M	Identifies the user connection.
SwitchFlag	M	This flag indicates whether switching is accepted.

Related: USwitchToVoiceModeInd.

5.6.6.4 Message parameters

This subclause describes parameters for each plane presented. They are alphabetically ordered. Table 60 only gives an overview of message parameters being User Plane protocol dependent.

Table 60: User Plane protocol dependent message parameters

Sub clause	Message Parameter Name	ETS 300 080 [1]	ISO 8208 [3]	ITU-T T.30 [14]
7.6.1	Algorithm		X	
7.6.2	Bcug	X	X	
7.6.4	Bit_DQM	see note	X	
7.6.5	CalledDTEAddress	X	X	X
7.6.6	CalledDTEAddressExt	X	X	
7.6.9	CallingDTEAddress	X	X	
7.6.10	CallingDTEAddressExt	X	X	
7.6.29	ExpeditedData		X	
7.6.31	FacilityData	see note	X	
7.6.32	FastSelect	see note	X	
7.6.33	GroupID	X	X	
7.6.42	L3ConnectionMode		X	
7.6.43	L3InComingVCCount		X	X
7.6.44	L3OutgoingVCCount		X	X
7.6.45	L3TwoWayVCCount	X	X	
7.6.52	PacketSize	see note	X	
7.6.54	QOSParameter	see note	X	
7.6.55	ReadyFlag	X	X	
7.6.58	RespondingDTEAddress	X	X	
7.6.59	RespondingDTEAddressExt	X	X	
7.6.67	WindowSize	see note	X	
7.6.70	X25Cause	X	X	
7.6.71	X25Diagnostics	X	X	
11.4.1	DataBlock			X
11.4.2	DataDescription			X
11.4.3	MailBoxMnemonic			X
11.4.4	MailBoxNumber			X
11.4.5	MailBoxType			X
11.4.6	NegociatedCharacteristic			X
11.4.8	PageAcknowledgement			X
11.4.9	Password			X
11.4.10	PollingNumber			X
11.4.11	PollingFlag			X
11.4.12	ReceivePageQuality			X
11.4.13	RemoteDesignation			X
11.4.14	SwitchFlag			X
11.4.15	T30Cause			X
NOTE: This parameter shall be used in accordance with the rules of ETS 300 080 [1].				

5.6.6.4.1 DataBlock

Description: This parameter is used to pass the transmission data to/from the PUF.

Type: .

Fields	Field type	Direction	Required	Comment
CurrentPage	Octet-string	B	M	The pages are numbered from 1 to n; n is the total of pages transmit. Length is fixed to 2.
ChainingFlag	Octet	B	M	The chaining flag indicates the position of the block in the page. bit 0: Last block of the page bit 1: First block of the page A block can be the first block and the last block of a page at the same time.
BlockNumber	Octet-string	B	O	For each page the blocks are numbered from 1 to p; p is the number of blocks needed to form the particular page. Length is fixed to 2.

Remark: Mandatory data shall be provided in data buffer; the value must not exceed the maximum value indicated in NAF-Property.

5.6.6.4.2 DataDescription

Description: This parameter is used to pass the characteristics of the transmission data to/from the PUF.

Type: x.

Fields	Field type	Direction	Required	Comment
PageNumber	Octet	B	M	If this parameter is equal to 0xFF, it is the end of the transmission of the document. In this case, the other parameters are not used
ResolutionUnit	Octet	B	M	Value for the resolution unit of measure: Default (255)- Default is Metric based (0) Metric based resolution (1) Inch based resolution
Resolution	Octet	B	M	Value for the resolution: Default (255)- Default is 200 x 200 dpi (0) R8 x 3,85 lines/mm (98 dpi) (1) R8 x 7,7 lines/mm (196 dpi) (2) 200 x 200 dpi (3) 300 x 300 dpi (4) 400 x 400 dpi (5) R8 x 15,4 lines/mm (392 dpi) (6) R16 x 15,4 lines/mm (392 dpi)
PageHeight	Octet	B	M	Values for page height: Default (255)- Default is A4 (0) A4 (1) B4 (2) Unlimited
PageWidth	Octet	B	M	Value for page width: Default (255)- Default is 1 728 pels (0) 1 728 pels/3 456 pels (1) 2 048 pels/4 096 pels (2) 2 432 pels/4 864 pels
CodingType	Octet	B	M	Values for the type of coding: Default (255)- Default is T4 (0) T4 (1) T4 bi-dimensional (2) T6 (3) BTM (Basic Transfer Mode) (4) DTM (Document Transfer Mode) (5) BFT (Binary File Transfer) (6) EDI (Edifact Transfer) (7) Character mode (8) Mixed mode (9) PM26
OctetPresentation	Octet	B	M	Value for the octet presentation: Default (255)- Default is octet not inverted (0) Octets not inverted in the page (1) Octets inverted in the page
PresenceFlag	Octet	B	M	Indicates the presence of the following fields: bit 0: MBoxMnemonic bit 1: MBoxPassword bit 2: DateTime

(continued)

Fields	Field type	Direction	Required	Comment
MBoxMnemonic	Octet-string	B	O	Used only if NAF can manage mail box. Length is fixed to 20.
MBoxPassword	Octet-string	B	O	Used only if NAF can manage mail box. Length is fixed to 20.
DateTime	Octet-string	B	O	This parameter is present: - during reception if the NAF has a timer - systematically during transmission if the PUF uses the strip service and if the NAF does not have a timer.
NOTE:	Resolutions of R8 and R16 are defined as follows: R8 = 1 728 pels/(215 mm ± 1%) for ISO A4 R8 = 2 048 pels/(255 mm ± 1%) for ISO B4 R8 = 2 432 pels/(303 mm ± 1%) for ISO A3 R16 = 3 456 pels/(215 mm ± 1%) for ISO A4 R16 = 4 096 pels/(255 mm ± 1%) for ISO B4 R16 = 4 864 pels/(303 mm ± 1%) for ISO A3			

5.6.6.4.3 MailBoxMnemonic

Description: This parameter is used to pass the mnemonic of the mail box.

Type: x.

Fields	Field type	Direction	Required	Comment
Mnemonic	Octet-string	P	M	Length is fixed to 20.

5.6.6.4.4 MailBoxNumber

Description: This parameter is used to pass the mail box identifier to/from the PUF.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	B	M	This value is unique for a PUF/NAF relation. Length is fixed to 20.

5.6.6.4.5 MailBoxType

Description: This parameter is used to pass the mail box type to the NAF.

Type: x.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	P	M	(1) used when receiving (2) used when polling

5.6.6.4.6 NegociatedCharacteristic

Description: This parameter is used to pass the characteristics of the transmission to the PUF.

Type: x.

Fields	Field type	Direction	Required	Comment
Speed	Octet	N	M	Default (255)- Default is 9 600 bits/s
				(1) 2 400 bits/s
				(2) 4 800 bits/s
				(3) 7 200 bits/s
				(4) 9 600 bits/s
				(5) 12 000 bits/s
				(6) 14 400 bits/s
				(7) 64 kbits/s
ECM	Boolean	N	M	Default (255)- Default is ECM used
				TRUE: ECM used
				FALSE: No ECM
ScanTime	Octet	N	M	Minimum scan line time:
				Default (255)- Default is 40 ms
				(1) 0 ms
				(2) 5 ms
				(3) 10 ms
				(4) 20 ms
				(5) 40 ms
PageNumber	Octet	N	M	
ResolutionUnit	Octet	B	M	Value for the resolution unit of measure:
				Default (255)- Default is Metric based
				(0) Metric based resolution
				(1) Inch based resolution
Resolution	Octet	B	M	Value for the resolution:
				Default (255)- Default is 200 x 200 dpi
				(0) R8 x 3,85 lines/mm (98 dpi)
				(1) R8 x 7,7 lines/mm (196 dpi)
				(2) 200 x 200 dpi
				(3) 300 x 300 dpi
				(4) 400 x 400 dpi
				(5) R8 x 15,4 lines/mm (392 dpi)
				(6) R16 x 15,4 lines/mm (392 dpi)

(continued)

Fields	Field type	Direction	Required	Comment
PageHeight	Octet	B	M	Values for page height: Default (255)- Default is A4 (0) A4 (1) B4 (2) Unlimited
PageWidth	Octet	B	M	Default (255)- Default is 1 728 pels (0) 1 728 pels/3 456 pels (1) 2 048 pels/4 096 pels (2) 2 432 pels/4 864 pels
CodingType	Octet	B	M	Values for the type of coding: Default (255)- Default is T4 (0) T4 (1) T4 bi-dimensional (2) T6 (3) BTM (Basic Transfer Mode) (4) DTM (Document Transfer Mode) (5) BFT (Binary File Transfer) (6) EDI (Edifact Transfer) (7) Character mode (8) Mixed mode (9) PM 26
MailBoxNumber	Octet	N	C	Only used on reception.

NOTE: Resolutions of R8 and R16 are defined as follows:
R8 = 1 728 pels/(215 mm ± 1%) for ISO A4
R8 = 2 048 pels/(255 mm ± 1%) for ISO B4
R8 = 2 432 pels/(303 mm ± 1%) for ISO A3
R16 = 3 456 pels/(215 mm ± 1%) for ISO A4
R16 = 4 096 pels/(255 mm ± 1%) for ISO B4
R16 = 4 864 pels/(303 mm ± 1%) for ISO A3

5.6.6.4.7 OctetInverted

Description: This parameter is used by the PUF to indicate if octets are inverted.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	(0) Octet not inverted (1) Octet inverted

5.6.6.4.8 PageAcknowledgement

Description: This parameter is used to indicate the PUF whether the remote station has received the current page.

Type: x.

Fields	Field type	Direction	Required	Comment
PageNumber	Octet	N	M	
RetransIndicator	Octet	N	M	Indicates whether the page must be retransmitted: (0) The page shall not be retransmitted (1) The page shall be retransmitted
Acknowledgement	Octet	N	M	Value for the acknowledgement of the remote station: (1) Page well received (2) Page badly received (3) Page not received (ECM only)

5.6.6.4.9 Password

Description: This parameter is used to pass the password of the mail box.

Type: x.

Fields	Field type	Direction	Required	Comment
Password	Octet-string	P	M	Length is fixed to 20.

5.6.6.4.10 PollingNumber

Description: This parameter is used to pass the number of polling authorised for a mail box used with polling.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	

5.6.6.4.11 PollingFlag

Description: This parameter is used to pass the result of polling request to the PUF or to indicate whether collection is accepted to the NAF.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	(0) Collection accepted (1) Collection refused (2) Remote station has documents to collect (3) Remote station has no document to collect or it refuses to collect them.

5.6.6.4.12 ReceivePageQuality

Description: This parameter is used to indicates whether the current page was correctly received.

Type: x.

Fields	Field type	Direction	Required	Comment
PageNumber	Octet	N	M	
QualityIndicator	Octet	N	M	(1) Page well received (2) Page poorly received (3) Page not received (ECM only)
ReceivedLine (Note)	Octet-string	N	C	Number of lines received. Length is fixed to 2
CorruptedLine (Note)	Octet-string	N	C	Number of corrupted lines. Length is fixed to 2
LastPageFlag (Note)	Octet	N	C	(0) Not the last page (1) Last page

NOTE: These parameters are only used in standard reception (not used in ECM).

5.6.6.4.13 RemoteDesignation

Description: This parameter is used to pass the identifier of the remote station and the contents of the non standard field to the PUF.

Type: x.

Fields	Field type	Direction	Required	Comment
Identifier	IA5-string	N	M	Remote station identifier. Length is fixed to 20
NSF_NSC	IA5-string	N	M	Specifies user requirements which are not covered by the ITU-TRec T.30 [14] . Maximum length is 40

5.6.6.4.14 SwitchFlag

Description: This parameter is used to pass the result of switching to voice request to the PUF or to indicate whether switching to voice is accepted to the NAF.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	(0) Switch accepted (1) Switch refused (2) Remote station has accepted the switch (3) Remote station has refused the switch

5.6.6.4.15 T30Cause

Description: This parameter is used to pass T.30 Cause to/from the PUF.

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	Value provided in table 60

Table 61: T30Cause value

Return Code		Meaning
DiscNorm	0	Disconnection normal
T1Exceeded	1	T1 exceeded
T2Exceeded	2	T2 exceeded
T5Exceeded	3	T5 exceeded
RemotePosInc	4	Remote possibility incompatible
FailTrain	5	Failure training
FailMessMode	6	Failure on entry in message mode
NonConform	7	Non conforming coding format in transmission
MaxScan	8	Maximum length of scan line exceeded
ReplyExceeded	9	Number of reply retransmission exceeded
ReplyNotExp	10	Reply not expected
RemoteDisc	11	Disconnection of remote
DataMiss	12	Absence of data transmission
RefusPolling	13	Polling refused
SubError	14	Sub address error (mail box does not exist)
PassError	15	Password error
CEDTimeOut	16	Time out on wait for CED

5.6.6.4.16 UseOfStrips

Description: This parameter is used by the PUF strip is use and when. Each information uses a binary position. The MSB is the bit 8 and the LSB is the bit 1. Bit 1 is for value 1, Bit 2 for value 2, Bit 2 for value 4 etc. The result value applying to this parameter is a sum of the value for each bit (logical OR).

Type: x.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	bit 0: On transmission bit 1: On reception bit 2: Data/Time display bit 3: Page numbering bit 4: ID display

5.7 Message parameters

This subclause describes parameters for messages of the Administration and the Control Plane. Parameters for messages of the User Plane are described in the User Plane clauses.

Parameters within this subclause are alphabetically ordered.

5.7.1 AdditionInformation

Description: This parameter is used to pass additional information from/to the PUF. The information is transparently conveyed to/from the network. See related network or NAF documentation for more details.

Type: 80.

Fields	Field type	Direction	Required	Comment
Value	IA5-string	B	M	128 is the maximum length. It may be reduced due to network constraints.

5.7.2 Algorithm

Description: This parameter passes the name of the security algorithm to be used to the NAF.

Type: 1.

Fields	Field type	Direction	Required	Comment
Algorithm	IA5-string	P	M	The security algorithm is identified by its name. The names of the available algorithms can be obtained using the Property information provided by the NAF. "nosecurity": this value for this parameter indicates that security is no longer needed for the connection. 16 bytes is the maximum length.

NOTE: A detailed description of the possible use of this parameter can be found in subclause 5.6.

5.7.3 BearerCap

Description: This parameter is used to pass bearer capability to/from the PUF and optionally layer 1 information if the LLC parameter, described in the subclause 5.7.37, is provided in the call.

Type: 3.

Fields	Field type	Direction	Required	Comment
BearerCap	Octet-string	B	M	Bearer capability information element. Maximum length is 12 octets.
NOTE: Values for this field are defined in the ETS 300 102 [2].				

5.7.4 CalledNumber

Description: This parameter is used to pass details concerning the called address to/from the PUF.

Type: 7.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	B	M	Default (255) - default is unknown unknown (0) international (1) national specific (2) network (3) subscriber (4) abbreviated (6)
NumberPlan	Octet	N	M	Default (255) - default is unknown unknown (0) isdn (1) data (3) telex (4) national (8) private (9)
Number	IA5-string	B	C	20 bytes is the maximum length. May be absent in the case of overlapping numbering or if associated with the CConnectReq message. Otherwise mandatory.
NOTE:	In the message exchange from PUF to NAF this parameter shall either be supplied in the NCO or in the appropriate message.			

5.7.5 CalledSubaddress

Description: This parameter is used to pass the Called Subaddress to/from the PUF.

Type: 8.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	B	M	Default (255) - default is nsap nsap (0) user (2)
Indicator	Octet	B	M	even(0) odd (1) This field is only meaningful if NumberType is set to user. It indicates if the number contains an odd or even number of BCD digits.
Number	IA5-string	B	M	20 bytes is the maximum length.

5.7.6 CallingNumber

Description: This parameter is used to pass details concerning the calling address to/from the PUF.

Type: 11.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	B	M	Default (255) - default is unknown unknown (0) international (1) national specific (2) network (3) subscriber (4) abbreviated (6)
NumberPlan	Octet	B	M	Default (255) - default is unknown unknown (0) isdn (1) data (3) telex (4) national (8) private (9)
Presentation	Octet	B	M	Default (255) - default is allowed allowed (0) restricted (1) not available (2) Indicates whether the Number should be provided to the called user
Screening	Octet	B	M	Default (255) - default is usernotscreened usernotscreened (0) userverified (1) networkprovided (3) Indicates any checking that has been applied to the Number
Number	IA5-string	B	M	20 is maximum length
NOTE:	Only "ISDN/telephony numbering plan" and "unknown" shall be allowed for the PUF as number plan identifier within the calling party number information element, when using Calling Line Identification Presentation supplementary service. Only "subscriber number", "national number" and "international number" shall be allowed for the PUF as type of number within the calling party number information element, when using Calling Line Identification Presentation supplementary service and specifying a complete number. Only "unknown" shall be allowed for the PUF as type of number within the calling party number information element, when using Calling Line Identification Presentation supplementary service and specifying an incomplete number for Direct Dialling In.			

5.7.7 CallingSubaddress

Description: This parameter is used to pass Calling Subaddress details to/from the PUF.

Type: 12.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	B	M	Default (255) - default is nsap nsap (0) user (2)
Indicator	Octet	B	M	even(0) odd (1) This field is only meaningful if NumberType is set to user. It indicates if the number contains an odd or even number of BCD digits.
Number	IA5-string	B	M	20 is maximum length

5.7.8 CAttributeName

Description: This parameter is used to pass the name of a static set of Control Plane attributes from the PUF.

Type: 13.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is the maximum length.

5.7.9 CauseToNAF

Description: This parameter is used to pass Cause Information from the PUF to the NAF.

Type: 14.

Fields	Field type	Direction	Required	Comment
Cause	Octet	P	M	Cause value.

5.7.10 CauseToPUF

Description: This parameter is used to pass Cause Information from the NAF to the PUF.

Type: 15.

Fields	Field type	Direction	Required	Comment
Cause	Octet	N	M	Cause value
Standard	Octet	N	M	Default (255) - default is ITU-T ITU-T (0) international (1) national (2) network (3)
Location	Octet	N	M	Default (255) - default is user user (0) privatelocal (1) publiclocal (2) transit (3) publicremote (4) privateremote (5) international (7) networkbeyond (10)
Recommendation	Octet	N	M	Default (255) - default is Q.931 Q.931 (0) X.21 (3) X.25 (4)
Diagnostics	Octet-string	N	C	Depends on the cause value. Length is fixed to 2. The lower octet contents least significant byte.

5.7.11 CDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the Control Plane part. If this parameter is absent at the NCO creation time, the value retained for this NCO will be both (3).

Type: 16.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	M	listen (1) call (2) both (3)

5.7.12 ChannelIdentification

Description: This parameter is used to pass Channel Information from/to the PUF.

Type: 17.

Fields	Field type	Direction	Required	Comment
Selection	Octet	B	M	nochannel (0) - no channel is available Bchannel (1) anychannel (3) - use any available channel Dchannel (4)
Number	Octet	B	O	This optional parameter is used by the PUF to select a particular B-channel. A value of 255 means select the first available B-channel.

Remarks: For CConnectReq message all values of Selection except nochannel and D-Channel are supported.

The number field can be used on the CAttribute set parameter structure or with CConnectReq message to select a particular permanent connected B-channel, or D-channel in the case where multiple TEIs are supported.

5.7.13 ChargingInfo

Description: This parameter is used to Transmit the charging information, if any, relevant to an NCO, in the Administration Attribute Set parameter.

Type: 18.

Fields	Field type	Direction	Required	Comment
Tag	Octet	N	M	charginginfo (3) chargingerror (4) (See subclause 5.7.32: coding of FacilityTag)
Value	Octet-string	N	C	Length and content depend on the Tag. Absent if Tag is chargingerror. (See subclause 5.7.32: coding of FacilityValue)

5.7.14 CompletionStatus

Description: This parameter is used to pass completion information to the PUF.

Type: 19.

Fields	Field type	Direction	Required	Comment
Status	Octet	N	M	Completion report value.
ErrorSpecific	Octet-string	N	C	Presence depends on value of Status field. Length shall be in the range 0 to 16 octets.

5.7.15 CongestionLevel

Description: This parameter is used to pass congestion level details to/from the PUF.

Type: 20.

Fields	Field type	Direction	Required	Comment
Level	Octet	B	M	ready (1) notready (15)

5.7.16 ConnectedNumber

Description: This parameter is used to pass details concerning the connected number to the PUF.

Type: 21.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	N	M	default (255) - default is unknown unknown (0) international (1) national (2) network (3) subscriber (4) abbreviated (6)
NumberPlan	Octet	N	M	default (255) - default is unknown unknown (0) isdn (1) data (3) telex (4) national (8) private (9)
Number	IA5-string	N	M	20 bytes is the maximum length

5.7.17 ConnectedSubaddress

Description: This parameter is used to pass the Connected Subaddress to the PUF.

Type: 22.

Fields	Field type	Direction	Required	Comment
NumberType	Octet	N	M	nsap (0) user (2)
Indicator	Octet	N	M	even(0) odd (1) This field is only meaningful if NumberType is set to user. It indicates if the number contains an odd or even number of BCD digits
Number	IA5-string	N	M	20 bytes is the maximum length

5.7.18 ControllerID

Description: This parameter allows the PUF to specify a particular controller. A ControllerID is based 0. In case of outgoing call, if a PUF does not specify a particular controller, the NAF applies its internal rules to make the selection, if multiple controller are available. In case of incoming calls, if no controller is indicated by the PUF as NCO parameter, the NAF will consider all available controllers as concern by the NCO.

Type: 83.

Fields	Field Type	Direction	Required	Comment
ControllerID	Octet	B	M	Controller number

5.7.19 CPMessageMask

Description: This parameter is set by the PUF to indicate which Control Plane messages are not intended to be received from the NAF. Not all Control Plane messages can be filtered.

This parameter is coded as a bit field. The default value is 0 for all bits, which means that the PUF will receive any message coming from the network.

Type: 73.

Fields	Field type	Direction	Required	Comment
CPMessageMask	Octet String	P	M	Fixed length is 2. Bit 1 CAlertInd Bit 2 CProgressInd Bit 3 CSetupAckInd Bit 4 CProceedingInd Bit 5 CUserInformationInd Bit 6 CCongestionControlInd Bit 7 CNotifyInd Bit 8 CFacilityInd Bits 9 to 16 are reserved

5.7.20 CPParameterMask

Description: This parameter is set by the PUF to indicate which Control Plane message parameters are not intended to be received from the NAF. Not all Control Plane message parameters can be filtered.

It is coded as a bit field. The default value is 0 for all bits, which means that the PUF will receive any Control Plane message parameter coming from the network.

Type: 72.

Fields	Field type	Direction	Required	Comment
CPParameterMask	Octet String	P	M	Fixed length is 4. Bit 1 CauseToPUF Bit 2 ChannelIdentification Bit 3 DateTime Bit 4 Display Bit 5 Facility Bit 6 High Layer Compatibility Bit 7 Low Layer Compatibility Bit 8 UserToUserInfo Bit 9 Signal Bit 10 ProgressIndicator Bits 11 to 31 are reserved

5.7.21 DateTime

Description: This parameter is used to pass date and time information to the PUF. This information is provided by the Network in the call establishment operation or by the NAF at the NCO creation time.

Type: 23.

Fields	Field type	Direction	Required	Comment
Year	Octet	N	M	0 to 99
Month	Octet	N	M	1 to 12
Day	Octet	N	M	1 to 31
Hour	Octet	N	M	0 to 23
Minute	Octet	N	M	0 to 59

5.7.22 Display

Description: This parameter is used to pass display information to the PUF.

Type: 24.

Fields	Field type	Direction	Required	Comment
Information	IA5-string	N	M	32 is maximum length

5.7.23 DtmfOperation

Description: This parameter is used to pass the information related the DTMF digits to send or to receive.

Type: 82.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	start listening DTMF digits (1) stop listening DTMF digits (2) send DTMF digits (3)

5.7.24 DtmfToneDuration

Description: This parameter is used to pass the information related the tone duration for one DTMF digit.

Type: 83.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	P	M	Time in ms (1 000 max.). Length is fixed to 2.

5.7.25 DtmfGapDuration

Description: This parameter is used to pass the information related the gap duration between 2 DTMF digits.

Type: 84.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	P	M	Time in ms (1 000 max.). Length is fixed to 2.

5.7.26 DtmfDigits

Description: This parameter is used to pass the information related the DTMF digits to send or received.

Type: 85.

Fields	Field type	Direction	Required	Comment
Digits	IA5-string	B	M	DTMF digits to send ('0' to '9') Numeric key. ('*') '*' key ('#') '#' key ('R') 'R.' key 32 is the maximum length. Each character generates an unique DTMF-tone

5.7.27 DtmfResult

Description: This parameter is used as the result of the acknowledgement of the CDtmfReq message sent by the PUF.

Type: 86.

Fields	Field type	Direction	Required	Comment
Dtmfconf	Boolean	N	M	True = Parameter OK False = One or more parameters are not accepted
Parameter_list	Octet	N	C	If Dtmfconf = False Type of parameter non accepted Bit 1 = 1 (DtmfOperation) Bit 2 = 1 (DtmfToneDuration) Bit 3 = 1 (DtmfGapDuration) Bit 4 = 1 (DtmfDigits) (See note)
NOTE	Each information uses a binary position. The MSB is the bit 8 and the LSB is the bit 1. Bit 1 is for value 1, bit 2 for value 2 and bit 3 for value 4. The result value applying to this parameter is a the sum of the value for each bit (logical OR).			

5.7.28 ExtEquipAvailability

Description: This parameter is used to pass the information related to the availability of the external equipment.

Type: 25.

Fields	Field type	Direction	Required	Comment
Availability	Boolean	N	M	State of the external equipment TRUE - equipment available FALSE - equipment unavailable

5.7.29 ExtEquipBlockDialling

Description: This parameter is used to pass the information related to the block dialling made with the keypad of the external equipment.

Type: 26.

Fields	Field type	Direction	Required	Comment
BlockDialling	IA5-string	N	M	Remote address and/or subaddress typed on the keypad of the external equipment. A star ('*') separates address and subaddress fields. 41 bytes is the maximum length.

5.7.30 ExtEquipKeyPressed

Description: This parameter is used to pass the information related to the pressed keys on the keypad of the external equipment.

Type: 27.

Fields	Field type	Direction	Required	Comment
Keypressed	Octet	N	M	Keypad information (0 to 9) Numeric key. (10) '*' key. (11) '#' key (12) 'A' key (13) 'B' key (14) 'C' key (15) 'D' key

5.7.31 ExtEquipName

Description: This parameter is used to pass the name that identifies an item of external equipment.

Type: 28.

Fields	Field type	Direction	Required	Comment
Type	Octet	B	M	type1 (1) - external equipment is of type 1. type2 (2) - external equipment is of type 2. type3 (3) - external equipment is of type 3. type4 (4) - external equipment is of type 4. type5 (5) - external equipment is of type 5. External equipment types are described in annex A.
Name	IA5-string	B	M	maximum length is 16 "DEFAULT" - use first defined external equipment of specified type

5.7.32 Facility

Description: This parameter is used to pass facility information to/from the PUF. If different facility information than defined in the FacilityTag 1 to 4 values is expected, the transparent (5) value should be used.

Type: 30.

Fields	Field type	Direction	Required	Comment
FacilityTag	Octet	B	M	chargingduring (1) - This value is used to request charging information during the connection. This tag is used in the direction from the PUF to the NAF. During the connection the NAF shall send subtotals of the charging information to the PUF. At the end of the connection the NAF shall provide the total charging information. chargingend (2) - This value is used to request charging information at the end of a connection. This tag is used in the direction from the PUF to the NAF. At the end of the connection the NAF shall provide the total charging information to the PUF. charginginfo (3)- This value is used to indicate that the Contents field contains charging related information. This value is used in the direction from NAF to PUF. chargingerror (4) - This value is used to indicate that the Contents field contains Error information related to the charging supplementary service. This value is used in the direction from NAF to PUF. transparent (5) - Allows the PUF or the NAF to send/receive facility information coded in format used by the network.
FacilityValue	Octet-string	B	C	Length and contents depend on value of FacilityTag field. Contents field is not allowed when FacilityTag field value is *chargingduring* or *chargingend.*

The coding of the FacilityValue field in the case when the FacilityTag field value is charginginfo, is defined in table 61.

NOTE: For transparent coding, the size of the contents is the size of the facility parameter minus 1.

Table 62: Coding of the FacilityValue field in the case of ChargingInfo

Subfield	Field type	Value	Comment
TypeOfTotal	Octet	subtotal (1)	Indicates whether the charging information is a total or a subtotal.
		total (2)	
TypeOfCharge	Octet	currencyinfo (2)	The charging information is represented as currency information.
		unitInfo (3)	The charging information is represented as charging units.
		freeofcharge (4)	The connection is free of charge.
		unknown (1)	The type of the charging information cannot be determined.
Value	Octet	value after decimal point. The value represents n x 1/256.	Value of the charging information in fixed point notation. If the octet 2 indicates freeofcharge, all three octets shall contain zero (0) to indicate a value of 00,0.
	Octet	least significant octet before decimal point	
	Octet	most significant octet before decimal point	

The coding of the Contents field in the case when the Tag field value is chargingerror, is defined in the table 63.

Table 63: Coding of the FacilityValue field in the case of ChargingError

Subfield	Field type	Value	Comment
ChargingError-Cause	Octet	notsubscribed (50)	The user has not subscribed to the Advice Of Charge (AOC) supplementary service.
		notavailable (63)	The AOC supplementary service is not available.
		notimplemented (69)	The AOC supplementary service is not implemented.
		InvalidCallState (101)	TheAOC supplementary service is invoked in an invalid call state. The supplementary service can be only be invoked in the CConnectReq.
		NoChargingInfoAvailable (128)	There is no charging information available.

5.7.33 GroupID

Description: This parameter is used to pass the group identifier to/from the PUF.

Type: 33.

Fields	Field type	Direction	Required	Comment
GroupID	Octet string	B	M	The value is unique for a PUF/NAF relation. 4 octets is the fixed length.

NOTE: A detailed description of the possible use of this parameter can be found in the User Plane clauses.

5.7.34 High Layer Compatibility (HLC)

Description: This parameter is used to pass HLC information to/from the PUF.

Type: 34.

Fields	Field type	Direction	Required	Comment
Standard	Octet	B	M	Default (255) - default is ITU-T ITU-T(0) international (1) national (2) network (3)
Identification	Octet	B	M	telephony (1) faxG4C1 (33) teletexF184 (36) teletexF220 (40) teletexF200 (49) videotext (50) telex (53) mhsx400 (56) osix200 (65) maintenance (94) management (95)
ExtIdentification	Octet	B	O	telephony (1) faxG4C1 (33) teletexF184 (36) teletexF220 (40) teletexF200 (49) videotext (50) telex (53) mhsx400 (56) osix200 (65)

5.7.35 Key

Description: Key to be used for the security algorithm.

Type: 36.

Fields	Field type	Direction	Required	Comment
Key	Octet-string	P	M	The Key parameter is used by the PUF to give relevant information for the security algorithm to the NAF. Maximum length is 255.

5.7.36 Keypad

Description: This parameter is used to pass keypad facility information to the NAF.

Type: 37.

Fields	Field type	Direction	Required	Comment
Keypad	Octet-string	P	M	IA5 characters to convey. Maximum length is 32.

5.7.37 Low Layer Compatibility

Description: This parameter is used to pass a subset of LLC information to/from the PUF. Information concerning layer 1 details shall be taken from the BearerCap parameter when an CConnectReq message is issued with an LLC parameter.

Type: 46.

Fields	Field type	Direction	Required	Comment
Negotiation	Boolean	B	M	TRUE - negotiation is allowed FALSE - negotiation is not allowed
Layer2protocol	Octet	B	M	0 - 31 255 = unspecified It refers to the Octet 6 of the LLC information element.
layer2optional	Octet	B	M	0 - 127 255 = unspecified It refers to the Octet 6a of the LLC information element
Layer3protocol	Octet	B	M	0 - 31 255 = unspecified It refers to the Octet 7 of the LLC information element
layer3optional	Octet	B	M	0 - 127 255 = unspecified It refers to the Octet 7a of the LLC information element

5.7.38 ManufacturerCode

Description: This parameter identifies the manufacturer. It is provided by the manufacturer.

Type: 47.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	B	M	Manufacturer identification. Maximum length is 255 octets.

5.7.39 NCOID

Description: This parameter is used to pass the connection object identifier to/from the PUF.

Type: 49.

Fields	Field type	Direction	Required	Comment
Value	Octet-string	B	M	This value is unique for a PUF/NAF relation. Length is fixed to 4 octets.

5.7.40 NCOType

Description: This parameter is used to pass the connection object type to the NAF.

Type: 50.

Fields	Field type	Direction	Required	Comment
Identifier	Octet	B	M	cset (1) - signalling access only. note
NOTE:	More values of the NCOType to be used in case of different types of User Plane Access can be found in the User Plane clauses.			

5.7.41 NotificationIndicator

Description: This parameter is used to pass notification of network event to the PUF. It may be a suspended or resumed operation.

Type: 51.

Fields	Field type	Direction	Required	Comment
Value	Octet	N	M	suspended (1) resumed (2) callwaiting (3)
NOTE:	Other values are network dependent.			

5.7.42 NumberComplete

Description: This parameter is used by the PUF to indicate to the NAF that a called number is complete.

Type: 79.

Fields	Field type	Direction	Required	Comment
Value	Octet	P	M	Number complete (1).
NOTE:	only one value is available.			

5.7.43 ProgressIndicator

Description: This parameter is used to pass information concerning the progress of a telephony call to the PUF.

Type: 53.

Fields	Field type	Direction	Required	Comment
Standard	Octet	N	M	ITU-T (ex-CCITT) (0) international (1) national (2) network (3)
Location	Octet	N	M	user (0) privatelocal (1) publiclocal (2) transit (3) publicremote (4) privateremote (5) international (7) networkbeyond (10)
Value	Octet	N	M	notISDN (1) - call is not end to end ISDN, further information may be available in-band. destinationnotISDN (2) - Destination address is not ISDN. originationnotISDN (3) - Origination address is not ISDN. returnedtoISDN (4) - Call has returned to ISDN. inbandinformation (8) - In-band information or appropriate pattern now available.

5.7.44 RequestID

Description: This parameter is used to pass an identifier to the NAF on a request message. It is returned by the NAF on the associated confirm message.

Type: 56.

Fields	Field type	Direction	Required	Comment
Identifier	Octet-string	B	M	Internal ID provided by the PUF. Length is fixed to 4 octets.

5.7.45 SelectorID

Description: This parameter is used by the PUF to select the right NCO on an incoming call (second step of the selection). Also the PUF uses the SelectorID to give the NAF a list of NCOs that should be exclusively dealt with.

Type: 60.

Fields	Field type	Direction	Required	Comment
Identifier	Octet-string	P	M	Internal ID provided by the PUF. Length is fixed to 4 octets.

5.7.46 Signal

Description: This parameter is used to optionally convey information to the PUF regarding tones and alerting signals.

Type: 81.

Fields	Field type	Direction	Required	Comment
Signal value	Octet	N	M	dial tone on (0) ring back tone on (1) intercept tone on (2) network congestion tone on(3) busy tone on (4) confirm tone on (5) answer tone on (6) call waiting tone on (7) off-hook warning tone on (8) tones off (63) alerting on - pattern 0 (64) note alerting on - pattern 1 (65) note alerting on - pattern 2 (66) note alerting on - pattern 3 (67) note alerting on - pattern 4 (68) note alerting on - pattern 5 (69) note alerting on - pattern 6 (70) note alerting on - pattern 7 (71) note alerting off (79)
NOTE: The use of these patterns is network dependent				

5.7.47 TEI

Description: This parameter is used to access a permanent link to a data packet switch (packet connection in D-channel).

Type: 61.

Fields	Field type	Direction	Required	Comment
Value	Octet	B	M	

5.7.48 UProtocol

Description: This parameter is used to select the User Plane protocol.

Type: 62.

Fields	Field type	Direction	Required	Comment
L3Protocol	Octet	P	M	Default (255). see note
L2Protocol	Octet	P	C	Mandatory if L3Protocol is NULL(4). Default (255) - ISO 7776 See note
L1Protocol	Octet	P	C	Mandatory if L2Protocol is NULL(8). Absent if L2Protocol is absent. see note.
NOTE:	A detailed description of the possible use and values for this parameter can be found in the User Plane clauses.			

5.7.49 UAttributeName

Description: This parameter is used to pass the name of a static set of User Plane attributes from the PUF.

Type: 63.

Fields	Field type	Direction	Required	Comment
AttributeName	IA5-string	P	M	16 bytes is maximum length.

5.7.50 UDirection

Description: This parameter is used to pass information concerning the usage of a particular NCO to the NAF, for the User Plane.

Type: 64.

Fields	Field type	Direction	Required	Comment
Direction	Octet	P	M	listen (1) call (2) both (3)
NOTE: if absent, the value assumed by the NAF is the value of the CDirection parameter.				

5.7.51 UserToUserInfo

Description: This parameter is used to pass user to user information to/from the PUF.

Type: 65.

Fields	Field type	Direction	Required	Comment
Discriminator	Octet	B	M	userspecific (0) - contents of information field is in user specific format. ia5chars (4) - contents of information field is IA5 characters.
Information	Octet-string	B	M	128 is maximum size.

Remarks: The Discriminator field is used to indicate the format of the data in the Information field. Values from 0 to 256 are possible but may be restricted by the ISDN being accessed. The values defined are supported by all NAFs.

5.7.52 AttributeSet Parameters

AttributeSet parameters depend on the type of parameters to provide with the ACreateNCO request. Tables 64 and 65 show the content of such parameters.

Table 64: Signalling Attribute Set (CAttributeSet) parameters

Parameters	Required	Comment
ChannelIdentification	O*	See subclause 5.7.12.
HLC	O	See subclause 5.7.34.
LLC	O	See subclause 5.7.37.
BearerCap	O	See subclause 5.7.3.

Table 65: External Equipment related parameters (within the UAttributeSet)

Parameters	Required	Comment
ExtEquipName	O	Name of external equipment to be used. If provided connection shall be established to identified external equipment and User Plane messages shall not be provided. See subclause 5.7.31.

Tables providing further detailed description of the User Plane Attribute Set (UAttributeSet) parameters available to be used with the ACreateNCO request for the various possible user protocols can be found in the User Plane clauses.

5.7.53 Administration AttributeSet parameters

Administration AttributeSet parameters are used to collect some management information about each NCO and are accessible at any time through the GetNCOInfo operation. Table 66 shows the contents of this parameter.

Table 66: Administration Attribute Set parameters

Parameters	Required	Comment
NCOType	O	See subclause 5.7.40.
CDirection	O	See subclause 5.7.11.
CAttributeName	O	See subclause 5.7.8.
CAttribute parameter	O	See table 64.
UDirection	O	See subclause 5.7.50.
UAttributeName	O	See the relevant User Plane subclauses.
UAttribute parameter	O	See the relevant User Plane subclauses.
CAddress parameters	O	See table 67.
UAddress parameters	O	See the relevant User Plane subclauses.
GroupID	O	Providing at the NCO creation time. See subclause 6.4.6.33.
SelectorID	O	See subclause 5.7.45.
ChargingInfo	O	See subclause 5.7.13.
DateTime	O	Date and Time of the NCO creation. See subclause 5.7.21.
CauseToPUF	O	See subclause 5.7.10.

5.7.54 AddressSet parameter

Tables 67 shows the structures of the Address.

Table 67: Signalling Address Set (CAddressSet) parameters

Parameters	Required	Comment
CalledNumber	O	See subclause 5.7.4 for parameter definition.
CalledSubaddress	O	See subclause 5.7.5 for parameter definition.
CallingNumber	O	See subclause 5.7.6 for parameter definition.
CallingSubaddress	O	See subclause 5.7.7 for parameter definition.

Tables providing further detailed description of the User Plane Address Set (UAddressSet) Parameters available to the PUF for the various possible user protocols can be found in the User Plane clauses.

5.8 Selection criteria

5.8.1 NCO Selection

In order to apply the right NCO on an incoming call, the following considerations have to be taken into account by the NAF.

Only the NCOs with UDirection or CDirection set to incoming or both directions are dealt with in this case. The best NCO shall contain an explicit definition for each value used for the checking. The "match" level shall be put on values checked rather than on values assumed. Table 68 summarises the matching operation.

Table 68: Matching operation for the NCO selection

Network	NCO	Operation	Result
Provided	Provided	Equal	Match
Provided	Provided	Not equal	No match
Provided	Not provided	(no operation)	Match
Not provided	Provided	(no operation)	No match
Not provided	Not provided	(no operation)	Match

The NAF shall broadcast an incoming call to all PUFs, which have indicated compatibility within a NCO. The incoming call shall then be assigned to the PUF which first accepts the call with the appropriate

message. All other PUFs shall receive a disconnect indication. Using this procedure also implies that NAF co-ordinated NCOs have a higher priority than PUF co-ordinated NCOs, since the NAF may respond immediately to an incoming call without involving any PUF. In such a case, the call is not seen from non-co-ordinated NCOs.

If CAlertReq message is sent by a PUF, only the first shall be sent to the network. All others are ignored. When a CDisconnectReq message is sent by a PUF, it does not disconnect the call except if no other NCO has been assigned to this call. This mechanism gives the opportunity to make connection with a delayed NCO.

The SelectorID parameter impacts on the incoming call broadcasting operation when more than one NCO per PUF is selected.

5.8.1.1 Control Plane information elements

1) Called Address (correct or absent);
2) Called Subaddress (correct or absent);
3) Bearer capabilities (correct);
4) LLC (see note) (correct or absent);
5) HLC (correct or absent).

These five information elements shall match the NCO values to make an NCO eligible. At the end of the selection process, if more than one NCO is eligible, the second step selection shall apply. First the check function, associated with the order of the information elements, shall be used to select a NCO. The latest selection criteria shall be the time. The latest NCO created by the NAF shall be selected first.

EXAMPLE: In the case presented in the table 69, only the NCO2 shall be chosen because the NCO1 is waiting for a subaddress information element different than provided in the incoming call and because all the expected information elements included in the NCO2 matches with the incoming call information elements.

Table 69: Matching NCO on an incoming call

Field	Incoming call	NCO1	NCO2
Called addr	123456789	not provided	123456789
Called sub addr	1002	1001	Not provided
Bearer cap.	speech	not provided	speech
LLC	no outband negotiation	not provided	not provided
HLC	telephony	telephony	telephony

User Plane protocol specific information elements to be considered during the NCO selection process are specified for the various possible user protocols in the User Plane clauses.

5.8.2 Action if no NCO available

5.8.2.1 Control Plane incoming call

A disconnection cause #88 "incompatible destination" is issued by the NAF.

5.8.2.2 User Plane incoming call

The disconnection procedure for the various possible user protocols are specified in the User Plane clauses.

5.9 Error checking and codes

This subclause deals with the error checking provided by the Profile A. Initially the error checking methods employed by each plane are described. Then the function return codes and error return codes for each plane are defined and described.

5.9.1 Administration Plane

For Administration Plane messages, almost all messages operate in Request/Confirm pairs; there are no Indicate/Response messages. Any error detected in a request message shall be notified in the related confirm message.

For Administration Plane messages any error detected shall prevent an operation from being performed and hence prevent a change of state.

Within the Administration Plane the AErrorInd message is used to indicate errors which are not covered by the protocols which support the Control Plane and User Plane messages. For example, this message is used to inform the PUF that an invalid NCOID has been specified on a message.

5.9.2 Control Plane

When mandatory parameters are missing, or a content error occurs in a mandatory parameter, or a parameter is unrecognised, the NAF indicates the error to the PUF as given in subclause 5.9.2.1 to 5.9.2.3.

5.9.2.1 Invalid state for message

CStatusInd, no change of state for connection.

5.9.2.2 Mandatory parameters

In case of mandatory parameters missing, mandatory parameters content error or unrecognised parameter, the NAF indicates this error to the PUF as follows:

- for CConnectReq the PUF is sent an CDisconnectInd;
- for CDisconnectReq the PUF is sent an CDisconnectCnf;
- for any other message the PUF is sent CStatusInd, no operation is performed, and no change of state occurs.

5.9.2.3 Optional Parameter Content Error

The message shall be processed as if the parameter were not present, CStatusInd is sent to the PUF indicating the parameter in error.

5.9.3 Errors in facility requests

Errors related to facility requests depend on the facility being requested. In the case of Advice of Charge supplementary service, errors are indicated by the use of a CFacilityInd message. The message that generated this error is processed as if there were no facility information present. Specific errors are defined in table 72. When a PUF uses facilities in the transparent form it is up to the PUF to understand how errors will be reported and what processing may have occurred within the network.

5.9.4 User Plane

Subclause 5.2.8 deals with the Error of Administration nature.

Errors are dealt with according to procedures defined in the User Plane clauses for the various possible User Plane protocols.

5.9.5 Function return codes

Table 70 defines function return codes:

Table 70: Function return codes

Return Code		Meaning
Success	0	Function completed successfully.
QueryEntityNotAvailable	128	The Query entity is not available or an error occurred during dialogue between the PUF and the Query entity.
InvalidSignalNumber	129	The signal number specified is invalid.
InvalidPCIHandle	130	Handle does not identify a NAF.
NAFnotAvailable	255	NAF is no longer available. The NAF has terminated due to error. This is a fixed condition.
NAFBusy	132	NAF is unable, currently, to process this request (lack of resource or other reason). The function may work correctly if reused at a later time. This is a temporary condition.
MaxPUFsExceeded	133	NAF can support no more PUFs.
InvalidPUFType	134	Invalid or unsupported type of PUF. NAF does not support this type of PUF.
InvalidPCIVersion	135	Invalid or unsupported version of PCI. NAF does not support this version of PCI.
InvalidExID	136	NAF does not recognise Exchange identifier.
InvalidPCIMPB	137	PCI Message Parameter Block address is incorrect.
InvalidMessageBuffer	138	Message Buffer address is invalid.
InvalidDataBuffer	139	Data Buffer address is invalid.
PCIMPBBufferTooSmall	140	PCIMPB Buffer is too small. Provided for operating systems that can check length of available memory.
MessageBufferTooSmall	141	Message Buffer is too small. Message Buffer does not meet message identifier requirements or actual buffer size in PCIMPB is greater than maximum size. On some operating systems this may also indicate that maximum size of data buffer exceeds memory limitations.
DataBufferRequired	142	Data Buffer is required for message.
DataBufferTooSmall	143	Data Buffer provided for message is too small. Data Buffer does not meet message identifier requirements or actual buffer size in PCIMPB is greater than maximum size. On some operating systems this may also indicate that maximum size of data buffer exceeds memory limitations.
PropertyBufferTooSmall	144	The buffer provided with the property information structure(s) is too small.
MessageTooLarge	145	There is no upper bound to the message size because of repetitions of parameters. If the message size exceeds the maximum size possible in an implementation, this value is returned.
InvalidHandlesBuffer	146	The PCIHandles buffer address is invalid.
HandlesBufferTooSmall	147	The size of the buffer for PCIHandles is too small to contain all available PCI_HANDLEs.
BufferTooSmall	148	The size of the buffer provided by the PUF is too small to answer the NAF needs (Operating System specific return code).
InvalidRegisterInfoStructure	149	At least one parameter contained in the PCIRegisterInfo structure is invalid (Operating System specific return code).
InvalidOpSysInfoStructure	150	At least one parameter contained in the PCIOpSysInfo structure is invalid (Operating System specific return code).

5.9.6 Administration Plane return code

The following values (see table 71) are returned in the CompletionStatus parameter The ErrorSpecific information column indicates what, if any, information shall be in the ErrorSpecific field:

Table 71: Administration Plane return code

Return Code		Meaning	ErrorSpecific Information
Success	0	Operation completed successfully.	Not present
NAFnotAvailable	255	NAF is no longer available. The NAF has terminated due to error. This is a permanent condition.	Not present
RessourceNotAvailable	47	Used with the NCO creation request message to indicate the lack of a resource (e.g. memory).	Not present
UndefinedMsgType	95	This message identifier is not defined by the Profile A.	Message Identifier
UnsupportedMsgType	97	This message identifier is defined by the Profile A but not supported by this NAF.	Message Identifier
InvalidParameter	99	A parameter is not recognised or is not supported by a message.	Parameter Type
MissingParameter	96	A mandatory parameter is missing from a message.	Parameter Type
InvalidParameterLength	182	A parameter's length is outside the allowed range for the parameter.	Parameter Type
InvalidContents	100	A parameter's content is invalid. Used with the NCO creation confirm message to report errors within parameters used to define the NCO.	Parameter Type
InvalidNCOID	81	A message has been passed to the NAF with an invalid NCOID.	NCOID value
NCOIDinUse	183	An NCOID that is in use for an established/establishing connection cannot be used on this message.	NCOID value
InvalidNCOType	184	A message has been passed to the NAF with an invalid NCOType value.	NCOType value
InvalidDirectionType	185	A message has been passed to the NAF with an invalid Direction value.	Not present
AttributeNameError	186	Invalid use of Attribute name. Name is not known, already defined or identifies an attribute set of the wrong type.	Attribute name
ExtraSetError	189	Message contains attribute set name that is not required.	Attribute name
SecurityNotActivated	190	Requested security algorithm has not been activated.	Security algorithm specific value.
InvalidCoordValue	191	Invalid value in NAF Co-ordination parameter.	Not present
InvalidGroupID	192	GroupID value is not recognised by the NAF.	GroupID value
GroupIDError	193	Message is either missing or requires a GroupID.	Not present
InvalidExtEquipName	194	External Equipment name is not known to NAF.	Not present
InvalidExtEquipType	195	Invalid value specified for External Equipment type.	Not present
OperationFailed	196	Requested operation failed.	Not present
ManufacturerCodeError	197	Error in the manufacturer code.	Specific manufacturer complement
FunctionalityNotProvided	198	Functionality not Provided by the NAF.	Not present.

5.9.7 Control Plane causes

These values are returned in the CauseToPUF parameter inside the "Cause" field when the parameter is part of a message passed from NAF to PUF.

NOTE: N/A means Not Applicable.

Table 72: Control Plane causes

Value	Q.931 [1] Meaning	Profile A Meaning	Generated by	NAF provided Diagnostics
1	Unallocated (unassigned) number		ISDN	N/A
2	No Route to Specified Transit Network		ISDN	N/A
3	No route to destination		ISDN	N/A
6	Channel Unacceptable		ISDN	N/A
7	Call placed on an already established channel		ISDN	N/A
16	Normal call clearing		ISDN	N/A
17	User busy		ISDN	N/A
18	No user responding		ISDN	N/A
19	No answer from user (user alerted)		ISDN	N/A
21	Call Rejected		ISDN	N/A
22	Address changed		ISDN	N/A
26	Non-selected user clearing		ISDN	N/A
27	Destination out of order		ISDN	N/A
28	Invalid address format	Parameter has invalid address format.	NAF, ISDN	Not present
29	Facility rejected	Facility is not provided by this NAF.	NAF, ISDN	Not present
30	Response to STATUS ENQUIRY		ISDN	N/A
31	Normal unspecified		ISDN	N/A
34	No circuit/channel available	Temporarily no channel of requested type is available from this NAF.	NAF, ISDN	Not present
41	Temporary Failure		ISDN	N/A
42	Switching equipment congestion		ISDN	N/A
43	Access information discarded	Parameter(s) information discarded.	NAF, ISDN	Parameter Types
44	Requested channel/circuit not available	No channel of requested type is available from this NAF.	NAF, ISDN	Not present
47	Resource unavailable, unspecified	Requested external equipment is not available.	NAF, ISDN	Not present
49	Quality of service unavailable		ISDN	N/A
50	Facility requested on Facility parameter is not subscribed		ISDN	N/A
57	Bearer Capability not authorised		ISDN	N/A
58	Bearer Capability not presently available		ISDN	N/A
63	Service or option not available, unspecified		ISDN	N/A
65	Service requested by Bearer Capability is not implemented		ISDN	N/A
66	Channel Type not implemented	NAF does not support this type of channel.	NAF, ISDN	Not present
69	Facility requested is not implemented	NAF does not support this facility.	NAF, ISDN	Not present
79	Service or option not implemented, unspecified		ISDN	N/A
81	Invalid call reference	Invalid NCOID.	NAF	Not present
82	Identified channel does not exist	Identified permanent channel is not defined.	NAF	Not present
85	No call suspended	NCOID does not identify a suspended connection.	NAF	Not present
88	Incompatible destination		ISDN	N/A
96	Mandatory parameter is missing	Mandatory parameter is missing.	NAF	Parameter Type
97	Message Identifier non-existent or not implemented on this network	Message Identifier non-existent or not implemented on this NAF.	NAF	Message Identifier
98	Message not compatible with call state or Message Identifier non-existent or not implemented.	Message not compatible with NCO state or Message Identifier non-existent or not implemented.	NAF	Message Identifier
99	Invalid parameter	Invalid parameter.	NAF	Parameter Type
100	Invalid parameter contents	Invalid parameter contents.	NAF	Parameter Type
101	Message not compatible with current state	Message not compatible with current state.	NAF	Message Identifier
127	Inter working, unspecified		ISDN	N/A

These values are valid in the CauseToNAF parameter "Cause" field when the parameter is part of a message passed from PUF to NAF. If an invalid value is used it shall be ignored and a value of 16, Normal call clearing, used in its place.

For some ISDN, other values may be provided in the NAF to PUF direction.

Table 73: Content of the CauseToNAF parameter

Value	Meaning
16	Normal call clearing.
21	Call rejected.
31	Normal unspecified.
88	Incompatible destination.

5.9.8 User Plane causes

Values returned as completion causes for messages of the User Plane can be found in the User Plane clauses.

5.10 Security

This subclause addresses communication security using Profile A.

5.10.1 General aspects of security in ISDN

The digital nature of ISDN facilitates adding security, but ISDN has been developed without support for security features in the lower layers. The deployment of ISDN in the public network is well under way and this constrains how security features may be added.

From the point of view of applications, the following needs for security can be seen:

- protecting information confidentiality;
- identifying the parties in communications (authentication);
- assuring the integrity of communicated information;
- controlling access to network services and customer equipment and data;
- being able to prove to a third party the fact that a communication occurred, the contents and the identities of the parties involved (non-repudiation).

From a security perspective, ISDN is more than a lower-layer communication service. Within the ISDN there has to be some concern for the applications, and especially the security requirements of these applications.

A foundation of common security standards for ISDN, particularly for authentication, confidentiality and integrity can provide the needed platform upon which the specific security needed by various ISDN applications can be built. The needed technology exists; it remains only to adopt it to ISDN and incorporate it in standards.

5.10.2 Security in Profile A

Although there are no lower layer ISDN security standards available, Profile A offers access to security functionalities which may be available in the NAF. This access offers an initial approach to use security on the ISDN.

The PUF can access the security functionality of the NAF in the following ways:

- using the supplementary service Calling Line Identification Presentation (CLIP);

  The CLIP supplementary service provides the PUF with the calling user's ISDN number, possibly with sub-address information. The ISDN number and sub-address information are provided by the network, and therefore, may be used to identify the calling user. This supplementary service provides the PUF with a method to identify the other party.

- Using the security messages in the Administration Plane:

  - ASecurityReq;
  - ASecurityCnf.

This security access provides the PUF access to encryption and security features which can be provided by the NAF. These messages provide a way to exchange the information needed for using the security features of the NAF. This security access provides a method for protecting information confidentiality. See subclauses 5.4.9 and 5.4.10 for information on the use of these Administration Plane messages.

5.10.3 Increasing security in Profile A

As no standards exist for security in ISDN, only limited features for security can be added in the Profile A. These security features are described in subclause 5.10.2.

Although the standards do not exist, the impact of introducing security in Profile A can be estimated. Three approaches to introducing security can be seen:

a) Security features as supplementary services

There should be little impact on Profile A interface or PUFs. These supplementary services should be handled in the same way as the normal ones.

b) Security as one specific protocol in the NAF

If on one of the lower layers a secure protocol is operating, the PUF may only have to supply this protocol with the necessary security information. This can be achieved by extending the Administration Plane to allow the transfer of the information. There are several ways to implement such extensions:

- adding a message;
- extending the attribute sets to contain the security information;
- NCOs contain the security information.

c) Definition of a new secure protocol stack for ISDN

If new secure ISDN protocols are established, Profile A shall be altered. New User Plane and Control Plane protocols need to be established. The extension mechanism provided by Profile A can be used to supply these new protocols.

Although the impact of introducing security can be estimated, the actual introduction of additional security features in Profile A is for further study.

6 Profile B

6.1 Readers guidance

Subclause 6.2 describes the basic mechanisms that ensure operating system independence such as messages, message structures and the used message protocol. Subclause 6.3 describes the operations necessary to exchange messages between Profile B and applications. Subclause 6.4 gives an overview of the messages in the Administration Plane, as does subclause 6.5 for the Control Plane messages and subclause 6.6 for the User Plane messages. Subclauses 6.7 and 6.8 specify in detail the functionality and coding of each message and parameter. Subclause 6.9 defines the allowed actions in different states of a connection by introducing a presentation of state diagrams in the form of the original Profile B document. The annex provides the state diagrams in standard SDL notation. Clause 7 includes all operating system dependent operations of Profile B to exchange messages. It is divided into subclauses for each operating system supported by Profile B. Annex L gives an intuitive understanding of how to connect, exchange data and disconnect, exemplified by arrow diagrams. Annex M provides a coding scheme used by an implementation of Profile B to exchange facsimile Group 3 documents between implementations of this Profile and applications. Annex N gives a short list of the protocols supported by Profile B. Annex P provides the state diagrams for Profile B conforming to ITU-T Recommendation Z.100 [5]. Annex Q provides an index listing each message, parameter and operation of this Profile. Finally, annex R acts as a guideline to the implementor of this Profile.

6.2 Message overview

The term *message* is a fundamental one to define Profile B. An asynchronous mechanism, used to exchange information defined by Profile B (*messages*), achieves operating system independence.

6.2.1 General message protocol

Communication between application and Profile B always uses the following general protocol:

A message shall be followed by a corresponding response. Messages from an application going to Profile B are called **REQUESTs**, the appropriate answer from Profile B is called **CONFIRMATION**. On the other side, messages originating from Profile B are called **INDICATIONs**, the corresponding reactions of an application are called **RESPONSEs**. This is also reflected in the naming convention of messages: every message name ends with the appropriate suffix (_REQ, _CONF, _IND, _RESP).

Each message contains a message number. Profile B shall return the number used in the REQUEST message in the corresponding CONFIRMATION. Applications may choose unique message numbers to identify message correlations before interpreting incoming messages. INDICATIONS from Profile B shall be numbered so that an application is guaranteed to get different message numbers for every incoming INDICATION.

An application shall not be allowed to send RESPONSE messages without receiving an INDICATION. Profile B shall ignore these illegal messages.

6.2.2 Type definitions

Parameters are associated with every message exchanged. To describe the message and its parameters, only a few basic types are used:

- **Byte:** coded as one octet;
- **Word:** coded as two contiguous octets, least significant first;
- **Dword:** coded as two contiguous words, least significant first;
- **Struct:** coded as an array of octets, the first octet containing the length of following data. If the first octet has the value **255** (0xFF), it indicates an escape character for interpreting the following word as containing the length of the data. An empty struct is coded as one single octet with value 0.

Every message is described in terms of these basic types.

6.2.3 Message structure

All messages exchanged between application and Profile B consist of a fixed-length header and a parameter area of variable length, parameter followed by parameter. No padding occurs in the message or parameter area.

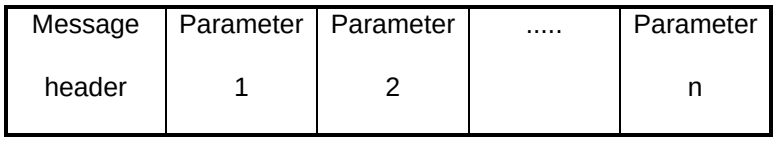


Figure 30: Message layout

In order to facilitate future extensions of this ETS, messages containing additional parameters shall be treated as valid messages. Profile B implementations and applications shall ignore all additional parameters.

The message header has the following layout:

Total length	ApplID	Command	Sub-command	Message number

Figure 31: Message header layout

Explanation of message header:

Message	Type	Contents
Total length	word	Total length of the message including the complete message header.
ApplID	word	Identification of the application. The application number is assigned to the application by Profile B in the CAPI_REGISTER operation
Command	byte	Command
Subcommand	byte	Command extension
Message number	word	Message number as described above

6.2.4 Manufacturer specific expansion

Manufacturer specific expansions of Profile B can be made without altering the basic structure. They are identified by an appropriate command/subcommand field in the message.

6.2.5 Table of messages

Messages are logically grouped into three types:

- messages concerning the signalling protocol of the ISDN (D-channel);
- messages concerning logical connections (B- or D-channel);
- administrative and other messages.

Tables 74, 75 and 76 give an overview of the defined messages and their functionality. The complete description of each message is given in subclause 6.7.

Table 74: Messages concerning signalling protocol

Message	Description
CONNECT_REQ	initiates an outgoing physical connection
CONNECT_CONF	local confirmation of request
CONNECT_IND	indicates an incoming physical connection
CONNECT_RESP	response to indication
CONNECT_ACTIVE_IND	indicates the activation of a physical connection
CONNECT_ACTIVE_RESP	response to indication
DISCONNECT_REQ	initiates clearing of a physical connection
DISCONNECT_CONF	local confirmation of request
DISCONNECT_IND	indicates the clearing of a physical connection
DISCONNECT_RESP	response to indication
ALERT_REQ	initiates sending of ALERT, i.e. compatibility to call
ALERT_CONF	local confirmation of request
INFO_REQ	initiates sending of signalling information
INFO_CONF	local confirmation of request
INFO_IND	indicates selected signalling information
INFO_RESP	response to indication

Table 75: Messages concerning logical connections

Message	Description
CONNECT_B3_REQ	initiates an outgoing logical connection
CONNECT_B3_CONF	local confirmation of request
CONNECT_B3_IND	indicates an incoming logical connection
CONNECT_B3_RESP	response to indication
CONNECT_B3_ACTIVE_IND	indicates the activation of a logical connection
CONNECT_B3_ACTIVE_RESP	response to indication
CONNECT_B3_T90_ACTIVE_IND	indicates switching from T.70NL to T.90NL
CONNECT_B3_T90_ACTIVE_RESP	response to indication
DISCONNECT_B3_REQ	initiates clearing of a logical connection
DISCONNECT_B3_CONF	local confirmation of request
DISCONNECT_B3_IND	indicates the clearing of a logical connection
DISCONNECT_B3_RESP	response to indication
DATA_B3_REQ	initiates sending of data on a logical connection
DATA_B3_CONF	local confirmation of request
DATA_B3_IND	indicates incoming data on a logical connection
DATA_B3_RESP	response to indication
RESET_B3_REQ	initiates the reset of a logical connection
RESET_B3_CONF	local confirmation of request
RESET_B3_IND	indicates the reset of a logical connection
RESET_B3_RESP	response to indication

Table 76: Administrative and other messages

Message	Description
LISTEN_REQ	activates call indications
LISTEN_CONF	local confirmation of request
FACILITY_REQ	requests additional facilities (e.g. ext. equipment)
FACILITY_CONF	local confirmation of request
FACILITY_IND	indicates additional facilities (e.g. ext. equipment)
FACILITY_RESP	response to indication
SELECT_B_PROTOCOL_REQ	selects current protocol stack of a logical connection
SELECT_B_PROTOCOL_CONF	local confirmation of request
MANUFACTURER_REQ	manufacturer specific operation
MANUFACTURER_CONF	manufacturer specific operation
MANUFACTURER_IND	manufacturer specific operation
MANUFACTURER_RESP	manufacturer specific operation

6.3 Exchange mechanism

6.3.1 Message queues

Communication between an application program and Profile B takes place via message queues. As shown in figure 32, there is exactly one message queue for Profile B and for each registered application program. Messages are exchanged between the applications programs and Profile B via these message queues. For data transfer the messages are used for control purposes only, and the data itself is transferred via a data area common to the application and Profile B. The queues are organised first in - first out, so Profile B shall process messages in the order of their arrival.

An application issues commands to an ISDN driver or controller by placing an appropriate message in the Profile B message queue. In the reverse direction, a message from an ISDN driver or controller is transferred to the message queue of the addressed application.

This method, used in higher-level protocols and modern operating systems, allows flexible access by several applications to different ISDN drivers and controllers. It also provides a powerful mechanism for processing events that arrive asynchronously, which is a paramount requirement for high speed data transfer.

The message queue structure is not specified. It is manufacturer-dependent and shall be transparent to the application program. The necessary access operations are defined by Profile B.

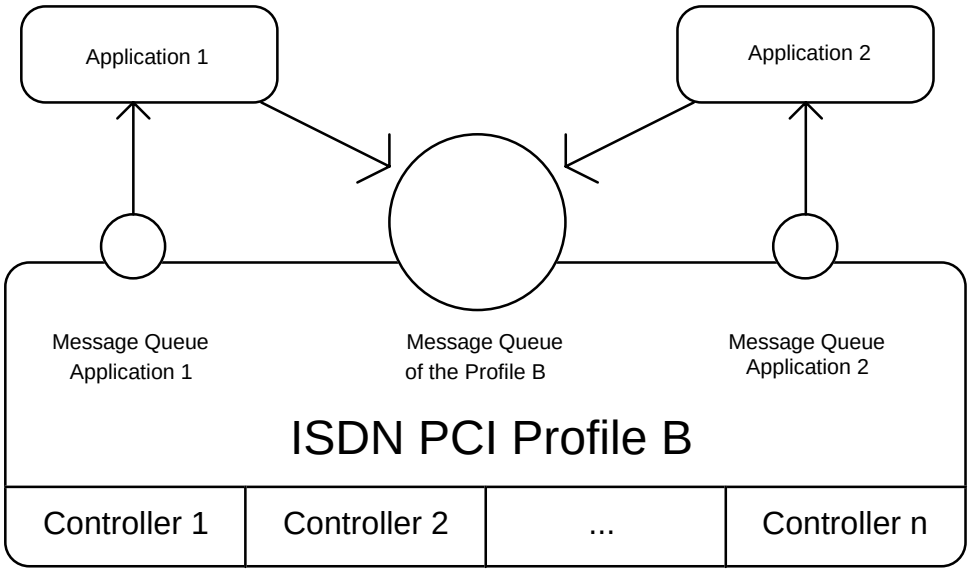


Figure 32: Message queues in Profile B

6.3.2 Operations on message queues

The message queues described represent the link between an application and Profile B with its connected ISDN drivers and controllers. Only four operations are required to use the message queues. The operations on the message queues are not restricted to a particular system specification. Their respective characteristics and implementation are operating system specific. At the same time, these operations form the complete interface which shall be matched to the particular operating system. The four operations are described below.

6.3.2.1 Registering an application

Before an application can issue commands to Profile B it shall register at Profile B. The CAPI_REGISTER function is used to do this. Profile B uses this function to assign a unique application number (ApplID) to the application. The message queue for the application is set up at the same time.

6.3.2.2 Messages from application to Profile B

All messages from an application to Profile B are put in the message queue of Profile B. The operation CAPI_PUT_MESSAGE is provided for this purpose. When this operation is used, the application transfers the message. If Profile B message queue cannot accept any more messages, the operation CAPI_PUT_MESSAGE returns an error.

6.3.2.3 Messages from Profile B to application

Profile B manages a message queue for each application; Profile B puts all messages to the application in this queue. The operation CAPI_GET_MESSAGE is provided for reading new messages from this queue. When this operation is used, it returns the received message. If the application message queue is empty, the operation CAPI_GET_MESSAGE returns an error. If an application does not retrieve these messages and message queue size configuration was too small, this queue may overflow. In this case, one or more messages from Profile B are lost. The application is informed of this error on the next CAPI_GET_MESSAGE operation.

6.3.2.4 Releasing an application

If a registered application wants to terminate Profile B usage, the connection to Profile B shall be released. This can be done with the CAPI_RELEASE operation. Releasing the application releases the previously used message queue. An application shall disconnect all existing connections before issuing a CAPI_RELEASE, otherwise the behaviour of Profile B is undefined. This is valid only for non-external equipment, external devices controlled by Profile B (e.g. phone) may allow releasing from Profile B without terminating existing calls.

6.3.2.5 Other operations

Additional operations are available to get information about manufacturer, software releases, configuration and serial numbers. Depending on the operating system there is also a possibility to register a call-back function which shall be activated if a new message is put in the application's message queue.

6.3.2.6 Manufacturer specific expansion

A manufacturer specific operation also exists, e.g. to configure ISDN controller.

6.3.3 Table of operations

Table 77: Operations defined in Profile B

Operation	Description
CAPI_REGISTER	Register an application
CAPI_RELEASE	Release an application
CAPI_PUT_MESSAGE	Transfer message to Profile B
CAPI_GET_MESSAGE	Get message from Profile B
CAPI_SET_SIGNAL	Register call-back function
CAPI_GET_MANUFACTURER	Get manufacturer identification
CAPI_GET_VERSION	Get Profile B version numbers
CAPI_GET_SERIAL_NUMBER	Get serial number
CAPI_GET_PROFILE	Get capabilities of Profile B implementation
CAPI_MANUFACTURER	Manufacturer specific function

6.4 Administration Plane

The following list gives an overview of the defined messages for the Administration Plane and their functionality. The complete description of each message is given in subclause 6.7.

Table 78: Administrative messages

Message	Description
LISTEN_REQ	activates call indications
LISTEN_CONF	local confirmation of request
FACILITY_REQ	requests additional facilities (e.g. ext. equipment)
FACILITY_CONF	local confirmation of request
FACILITY_IND	indicates additional facilities (e.g. ext. equipment)
FACILITY_RESP	response to indication
SELECT_B_PROTOCOL_REQ	selects current protocol stack of a logical connection
SELECT_B_PROTOCOL_CONF	local confirmation of request

6.5 Control Plane

Table 79 gives an overview of the defined messages for the Control Plane and their functionality. The complete description of each message is given in subclause 6.7.

Table 79: Messages concerning Control Plane

Message	Description
CONNECT_REQ	initiates an outgoing physical connection
CONNECT_CONF	local confirmation of request
CONNECT_IND	indicates an incoming physical connection
CONNECT_RESP	response to indication
CONNECT_ACTIVE_IND	indicates the activation of a physical connection
CONNECT_ACTIVE_RESP	response to indication
DISCONNECT_REQ	initiates clearing of a physical connection
DISCONNECT_CONF	local confirmation of request
DISCONNECT_IND	indicates the clearing of a physical connection
DISCONNECT_RESP	response to indication
ALERT_REQ	initiates sending of ALERT, i.e. compatibility to call
ALERT_CONF	local confirmation of request
INFO_REQ	initiates sending of signalling information
INFO_CONF	local confirmation of request
INFO_IND	indicates selected signalling information
INFO_RESP	response to indication

6.6 User Plane

Table 80 gives an overview of the defined messages for the User Plane and their functionality. The complete description of each message is given in subclause 6.7.

Table 80: Messages concerning logical connections (User Plane) and manufacturer messages

Message	Description
CONNECT_B3_REQ	initiates an outgoing logical connection
CONNECT_B3_CONF	local confirmation of request
CONNECT_B3_IND	indicates an incoming logical connection
CONNECT_B3_RESP	response to indication
CONNECT_B3_ACTIVE_IND	indicates the activation of a logical connection
CONNECT_B3_ACTIVE_RESP	response to indication
CONNECT_B3_T90_ACTIVE_IND	indicates switching from T.70NL to T.90NL
CONNECT_B3_T90_ACTIVE_RESP	response to indication
DISCONNECT_B3_REQ	initiates clearing of a logical connection
DISCONNECT_B3_CONF	local confirmation of request
DISCONNECT_B3_IND	indicates the clearing of a logical connection
DISCONNECT_B3_RESP	response to indication
DATA_B3_REQ	initiates sending of data on a logical connection
DATA_B3_CONF	local confirmation of request
DATA_B3_IND	indicates incoming data on a logical connection
DATA_B3_RESP	response to indication
RESET_B3_REQ	initiates the reset of a logical connection
RESET_B3_CONF	local confirmation of request
RESET_B3_IND	indicates the reset of a logical connection
RESET_B3_RESP	response to indication
MANUFACTURER_REQ	manufacturer specific operation
MANUFACTURER_CONF	manufacturer specific operation
MANUFACTURER_IND	manufacturer specific operation
MANUFACTURER_RESP	manufacturer specific operation

6.7 Message descriptions

The following subclause defines all Profile B messages with their respective parameters. Parameters are explained in more detail in this subclause.

Messages are sorted alphabetically irrespective of the extension, which defines the originator and direction of the message. The following order is used for basic names: REQUEST, CONFIRMATION, INDICATION, RESPONSE.

6.7.1 ALERT_REQ

Description

This message should be used by applications to indicate compatibility to an incoming call. It shall send an ALERT to the network to prevent the call from expiring (no user responding). If an application is able to accept the call immediately it shall not be necessary to use this message; the application can issue immediately a CONNECT_RESP to Profile B.

ALERT_REQ	Command	**0x01**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Additional info	struct	Additional info elements
NOTE: The parameter *Additional info* shall be coded as an empty structure if no additional information (e.g. user data) has to be transmitted.		

6.7.2 ALERT_CONF

Description

This message confirms the reception of an ALERT_REQ.

ALERT_CONF	Command	**0x01**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: alert initiated 0x0003: alert already sent by another application 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding
NOTE: Info 0x0003 shall be returned if another application already initiated the sending of an ALERT message to the network. In this case the parameter *Additional info* of the corresponding REQUEST has been ignored.		

See also: Description of broadcast mechanism in LISTEN_REQ

6.7.3 CONNECT_REQ

Description

This message initiates the set-up of a physical connection. An application may only offer the relevant parts of the parameters, i.e. *Controller*, *CIP Value, B protocol* and normally c*alled party number*. Every other structure can be empty (length of 0). In this case the default values as described in clause 6 shall be used.

CONNECT_REQ	Command	**0x02**
	Subcommand	**0x80**

Parameter	**Type**	**Comment**
Controller	dword	
CIP Value	word	Compatibility Information Profile
Called party number	struct	Called party number
Calling party number	struct	Calling party number
Called party subaddress	struct	Called party subaddress
Calling party subaddress	struct	Calling party subaddress
B protocol	struct	B protocol to be used
BC	struct	Bearer Capability
LLC	struct	Low Layer Compatibility
HLC	struct	High Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: If an application offers *BC*, *LLC* and/or *HLC*, the parameter shall be used without checking the resulting combination.
The absence (i.e. coding as an empty structure) of *B protocol* shall result in the default protocol behaviour: ISO 7776 [4] (X.75) and window size 7. This is a recommended selection to get overall connectivity with the benefits of HDLC error recovery. Note that ISO 7776 [4] deals with a default maximum data length of 128 octets, whereas Profile B is able to handle up to at least 2 048 octets, depending on **CAPI_REGISTER** values of an application.

6.7.4 CONNECT_CONF

Description

This message confirms the initiation of a call set-up. This connection is assigned a *PLCI* which serves as an identifier in further processing. Errors are returned in the parameter *info*.

CONNECT_CONF	Command	**0x02**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: connect initiated 0x2002: illegal controller 0x2003: out of PLCI 0x2007: illegal message parameter coding 0x3001: B1 protocol not supported 0x3002: B2 protocol not supported 0x3003: B3 protocol not supported 0x3004: B1 protocol parameter not supported 0x3005: B2 protocol parameter not supported 0x3006: B3 protocol parameter not supported 0x3007: B protocol combination not supported 0x300A: CIP Value unknown

NOTE: The connection is in the set-up phase at this point in time. Subsequent successful switching is indicated by the message **CONNECT_ACTIVE_IND**.
If an application wants to identify the corresponding REQUEST to this message, it may use the message number mechanism described in subclause 6.2.

6.7.5 CONNECT_IND

Description

This message indicates an incoming call for a physical connection. For the incoming call a PLCI is assigned which is used to identify this connection in subsequent messages.

CONNECT_IND	Command	**0x02**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
CIP Value	word	Compatibility Information Profile
Called party number	struct	Called party number
Calling party number	struct	Calling party number
Called party subaddress	struct	Called party subaddress
Calling party subaddress	struct	Calling party subaddress
BC	struct	Bearer compatibility
LLC	struct	Low Layer Compatibility
HLC	struct	High Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: To activate the signalling of incoming calls, the message **LISTEN_REQ** shall be send to the controller.
Every information available from the network at this point shall be signalled to the application. Empty structs show the absence of this information.

6.7.6 CONNECT_RESP

Description

This message is used to accept or reject an incoming call on behalf of the application. The incoming call is identified via parameter *PLCI*. The parameter *reject* is used to accept, reject or ignore the call. In case of ignoring the call, other ISDN equipment connected on the same bus (basic access) shall have the chance to accept this call, whereas the rejection of this incoming call should terminate the call on the entire bus. For primary access, these parameter values of parameter *Reject* shall behave identically.

CONNECT_RESP	Command	**0x02**
	Subcommand	**0x83**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Reject	word	0: accept call 1: ignore call 2: reject call, normal call clearing 3: reject call, user busy 4: reject call, requestet circuit/channel not available 5: reject call, facility rejected 6: reject call, channel unacceptable 7: reject call, incompatible destination 8: reject call, destination out of order
B protocol	struct	B protocol to be used
Connected number	struct	Connected number
Connected subaddress	struct	Connected subaddress
LLC	struct	Low Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: The parameter *LLC* can optionally be used for LLC negotiation if supported by the network.
Any unknown *reject* value shall be mapped to *normal call clearing*.
Any *reject* value other than *accept call* shall sent a DISCONNECT_IND to the application.
The absence (i.e. coding as an empty structure) of *B protocol* shall result in the default protocol behaviour: ISO 7776 [4] (X.75) and window size 7. This is a recommended selection to get overall connectivity with the benefits of HDLC error recovery. Note that ISO 7776 [4] deals with a default maximum data length of 128 octets, whereas Profile B is able to handle up to at least 2 048 octets, depending on **CAPI_REGISTER** values of an application.

6.7.7 CONNECT_ACTIVE_IND

Description

This message indicates the physical connection of a B-channel. The connection is identified by the parameter *PLCI*.

CONNECT_ACTIVE_IND	Command	**0x03**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Connected number	struct	Connected number
Connected subaddress	struct	Connected subaddress
LLC	struct	Low Layer Compatibility
NOTE: The parameter *connected number/subaddress* and *LLC* shall be filled in completely if this information is provided by the network. The absence of network information shall be indicated by empty structures.		

6.7.8 CONNECT_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_ACTIVE_IND**.

CONNECT_ACTIVE_RESP	Command	**0x03**
	Subcommand	**0x83**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier

6.7.9 CONNECT_B3_ACTIVE_IND

Description

This message indicates the logical connection of a B-channel. The connection is identified by the parameter *NCCI*. The parameter *NCPI* is used to transfer additional protocol dependent information.

CONNECT_B3_ACTIVE_IND	Command	**0x83**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
After this message incoming data can be indicated to the application.
In case of protocol T.30 [14] and outgoing calls, this message does not imply the successful training between both facsimile stations. This is to enable an application to send data to Profile B without waiting for termination of training phase. If this training phase is not successful, corresponding indications shall be given by Profile B in the message DISCONNECT_B3_IND.

6.7.10 CONNECT_B3_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_B3_ACTIVE_IND**.

CONNECT_B3_ACTIVE_RESP	Command	**0x83**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier

6.7.11 CONNECT_B3_REQ

Description

This message initiates the set-up of a logical connection. The physical connection is identified by the parameter *PLCI*. Additional protocol dependent information can be transferred with the parameter *NCPI*.

CONNECT_B3_REQ	Command	**0x82**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used.		

6.7.12 CONNECT_B3_CONF

Description

With this message the initiation of a logical connection set-up is confirmed. This connection is assigned a *NCCI*, which subsequently identifies this logical connection. Errors are supplied in the parameter *info*.

CONNECT_B3_CONF	Command	**0x82**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: connect initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2004: out of NCCI 0x3008: NCPI not supported
NOTE: The connection is in the set-up phase at this stage. The successful set-up shall be indicated by the message **CONNECT_B3_ACTIVE_IND**. If parameter *info* returns **0x0001**, the set-up of a logical connection is initiated, but parameter *NCPI* has been ignored. In that case the used layer 3 protocol does not support the usage of *NCPI* (e.g. the transparent mode of layer 3).		

6.7.13 CONNECT_B3_IND

Description

This message indicates an incoming call for a logical connection. For this incoming call a *NCCI* is assigned, which subsequently identifies the call. Additional protocol dependent information shall be transferred with parameter *NCPI* if available.

CONNECT_B3_IND	Command	**0x82**
	Subcommand	**0x82**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. The connection is in the set-up phase at this stage. The successful set-up shall be indicated by the message **CONNECT_B3_ACTIVE_IND**.		

6.7.14 CONNECT_B3_RESP

Description

With this message the application accepts or rejects an incoming logical call. The incoming call is identified via the parameter *NCCI*. The call can be accepted or rejected via the parameter *reject*. The parameter *NCPI* can be used to transfer additional protocol dependent information.

CONNECT_B3_RESP	Command	**0x82**
	Subcommand	**0x83**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier
Reject	word	0: accept call 2: reject call, normal call clearing
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. Any other value of *reject* shall result in rejecting the call.		

6.7.15 CONNECT_B3_T90_ACTIVE_IND

Description

This message indicates the switching from T.70 [15] to T.90 [16] within a logical connection of a B-channel. The connection is identified by the parameter *NCCI*. The parameter *NCPI* is used to transfer additional T.90 [16] information.

CONNECT_B3_T90_ACTIVE_IND	Command	**0x88**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: This message shall only be generated if the selected protocol is T.90NL with compatibility to T.70NL according to T.90 [16]Appendix II. In this case the initially used protool is T.70 [15]. This message indicates the negotiation and switching to T.90 [16].

6.7.16 CONNECT_B3_T90_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_B3_T90_ACTIVE_IND**.

CONNECT_B3_T90_ACTIVE_RESP	Command	**0x88**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier

6.7.17 DATA_B3_REQ

Description

This message sends data within the logical connection identified by the *NCCI*. Data to be sent is referenced via the parameter *data*/*data length*. The data is not part of the message, a 32-bit pointer is used to transfer the address of the data area. The application issues a unique identifier for this data in the parameter *data handle*. On subsequent confirmation by a **DATA_B3_CONF** this handle is used. It is possible to set additional information, such as more data, delivery confirmation etc. via parameter *flags*. The flags are not supported by all protocols.

DATA_B3_REQ	Command	**0x86**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data	dword	Pointer to the data to be sent
Data length	word	Size of data area to be sent
Data handle	word	Referenced in DATA_B3_CONF
Flags	word	[0]: qualifier bit [1]: more data bit [2]: delivery confirmation bit [3]: expedited data [4] to [15]: reserved

NOTE: The data transfer does not support assembly or re-assembly of data.
An application shall not change or free the data area until the corresponding **DATA_B3_CONF** is received.
Flags are protocol dependent. If an application set reserved bits in parameter *Flags*, Profile B shall reject the **DATA_B3_REQ**. This is to allow future expansion of this parameter. If an application set bits in parameter *Flags*, which are not supported by the current protocol, Profile B shall accept the **DATA_B3_REQ** but shall return this information in the corresponding **DATA_B3_CONF**.

6.7.18 DATA_B3_CONF

Description

This message confirms the acceptance of a data package to be sent. The logical connection is identified by the parameter *NCCI*. The parameter *data handle* supplies the identifier used by the application in the associated **DATA_B3_REQ** as reference to the transferred data area. After receiving this message, the application can reuse the referenced data area.

DATA_B3_CONF	Command	**0x86**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data handle	word	Identifies the data area of corresponding DATA_B3_REQ
Info	word	0: data transmission initiated 0x0002: flags not supported by current protocol, flags ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x300A: flags not supported (reserved bits) 0x300C: data length not supported by current protocol

NOTE: Every **DATA_B3_REQ** shall result in a corresponding **DATA_B3_CONF** except in the following case: after transmitting the message **DISCONNECT_B3_IND** to an application, Profile B is not allowed to send any other message concerning this logical connection identified by the parameter NCCI. So in this case the application shall assure that resources or buffer management will be reset correctly.
If an application sets the delivery confirmation bit in the corresponding DATA_B3_REQ and the selected protocol supports this mechanism the confirmation shall be given to the application after the delivery of the sent packet is confirmed by the used protocol.
Seven unconfirmed DATA_B3_REQ messages shall be supported.

6.7.19 DATA_B3_IND

Description

This message indicates incoming data within a logical connection. The logical connection is identified via the *NCCI*. The length of the incoming data area is indicated via the parameter *data length*. The incoming data area can be referenced by the parameter *data*. The data is not part of the message, a 32-bit pointer is used to transfer the address of the data area. Profile B issues a handle to this data area via the parameter *data handle*. On subsequent confirmation by a **DATA_B3_RESP**, this handle shall also be supplied by the application. Additional information, such as more data, delivery confirmation etc. is supplied by parameter *flags*, if available.

DATA_B3_IND	Command	**0x86**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data	dword	Pointer to data received
Data length	word	Size of data area received
Data handle	word	handle to data area, referenced in DATA_B3_RESP
Flags	word	[0]: qualifier bit [1]: more-data bit [2]: delivery confirmation bit [3]: expedited data [4 to 14]: reserved [15]: framing error bit, data may be invalid (only with corresponding B2 protocol)

NOTE: The data transfer does not support re-assembly functions.
The data area which contains the data remains allocated until the corresponding **DATA_B3_RESP** is received. However, expedited data is only valid until the next **CAPI_GET_MESSAGE** is performed by the application.
In case of receiving **DATA_B3_IND** messages with reserved bits switched on in the flags parameter an application shall ignore the data area but process the message, i.e. send a **DATA_B3_RESP** to Profile B. This is to allow future expansion of the *flags* parameter.

6.7.20 DATA_B3_RESP

Description

With this message the application confirms acceptance of an incoming data package. The logical connection is identified by the parameter *NCCI*. The parameter *data handle* identifies the data handle used by Profile B in the corresponding **DATA_B3_IND** as the reference to the transferred data area.

DATA_B3_RESP	Command	**0x86**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data handle	word	Data area reference in corresponding DATA_B3_IND

NOTE: This message frees the data buffer referenced by *Data handle* for reuse by Profile B.
Data throughput depends on an application's rapid response to DATA_B3_IND messages. Failure to do so shall trigger flow control on the line (for protocols supporting flow control such as ISO 7776 [4](X.75) or ISO 8208 [3](X.25)) and may cause loss of incoming data for protocols without flow control mechanism.

6.7.21 DISCONNECT_B3_REQ

Description

This message initiates the clearing of a logical connection identified via the parameter *NCCI*. The parameter *NCPI* can be used to transfer additional protocol dependent information.

DISCONNECT_B3_REQ	Command	**0x84**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
In case of facsimile group 3 (B protocol T.30 [14]) and speech (B1 protocol bit transparent, B2/B3 protocol transparent) data already given to transmission via **DATA_B3_REQ** shall be sent before disconnecting the logical connection.

6.7.22 DISCONNECT_B3_CONF

Description

With this message the initiation of clearing a logical connection is confirmed. Any errors are coded in the parameter *info*.

DISCONNECT_B3_CONF	Command	**0x84**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: disconnect initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x3008: NCPI not supported

6.7.23 DISCONNECT_B3_IND

Description

This message indicates the clearing of a logical connection identified via the parameter *NCCI*. The parameter *Reason_B3* indicates if this clearing is caused by wrong protocol behaviour. The parameter *NCPI* is used to indicate additional protocol dependent information if available.

DISCONNECT_B3_IND	Command	**0x84**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Reason_B3	word	0: clearing according to protocol 0x3301: protocol error layer 1 0x3302: protocol error layer 2 0x3303: protocol error layer 3 protocol dependent values are described in subclause 6.8
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the *NCPI* parameter depends on the protocol used.
After this message no other message concerning this *NCCI* shall be sent to the application. The application shall answer this message with **DISCONNECT_B3_RESP** to free the resources allocated to the *NCCI*.

6.7.24 DISCONNECT_B3_RESP

Description

With this message the application confirms the clearing of a logical connection.

DISCONNECT_B3_RESP	Command	**0x84**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NOTE: With this message resources allocated to the *NCCI* are released. If an application fails to send this message after receiving **DISCONNECT_B3_IND**, Profile B may reject subsequent **CONNECT_B3_REQ** with the info value **out of NCCI** (0x2004).		

6.7.25 DISCONNECT_REQ

Description

This message initiates the clearing of a physical connection, identified by the parameter *PLCI*.

DISCONNECT_REQ	Command	**0x04**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Additional Info	struct	Additional information elements
NOTE: Existing logical connections shall be cleared by Profile B using the message DISCONNECT_B3_IND containing the cause **protocol error layer 1** (0x3301) before clearing the physical connection.		

6.7.26 DISCONNECT_CONF

Description

This message confirms the initiation of clearing a physical connection. Any errors are coded in the parameter *info*.

DISCONNECT_CONF	Command	**0x04**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: disconnect initiated 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding

6.7.27 DISCONNECT_IND

Description

This message indicates the clearing of the physical channel identified via the parameter *PLCI*. The parameter *reason* indicates the network delivered cause or if this clearing is caused by wrong protocol behaviour

DISCONNECT_IND	Command	**0x04**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Reason	word	0: no cause available 0x3301: protocol error layer 1 0x3302: protocol error layer 2 0x3303: protocol error layer 3 0x3304: another application got that call 0x34xx: disconnect cause from the network according to Q.931 [8]/ETS 300 102-1 [2]. In the field 'xx' the cause value received within a cause information element (octet 4) from the network is indicated.
NOTE: After this message no other message concerning this *PLCI* shall be sent to the application. The application shall answer this message with **DISCONNECT_RESP** to free the resources allocated to the *PLCI*.		

6.7.28 DISCONNECT_RESP

Description

With this message the application confirms the clearing of the physical channel.

DISCONNECT_RESP	Command	**0x04**
	Subcommand	**0x83**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
NOTE: With this message the *PLCI* is released. If an application fails to send this message after receiving **DISCONNECT_IND** resources bound to this *PLCI* shall not be freed. This may lead to Profile B resource problems (indicated by info value **out of PLCI),** affecting other applications too.		

6.7.29 FACILITY_REQ

Description

This message is used to handle optional facilities on the *controller* or facilities related on connections identified by *PLCI* or *NCCI*. The struct *facility request parameters* is defined for each facility. At the moment facilities Handset Support and DTMF are defined. Handset Support is used to support external ISDN equipment, DTMF is used in the Public Switched Telephone Network (PSTN) to select and control several provided services (e.g. automatic answering service).

Handset Support as well as DTMF support are optional Profile B features. In case Profile B does not support these facilities, an appropriate information value is returned in the **FACILITY_CONF**.

DTMF cannot be used with all B protocols. Normally, it is used with 64 kbit/s bit speech and T.30 [14] audio.

FACILITY_REQ	Command	**0x80**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility request parameter	struct	Facility depending parameters

6.7.30 FACILITY_CONF

Description

This message confirms the acceptance of the **FACILITY_REQ**. The event is identified by *Controller/PLCI/NCCI*, depending on the facility. The struct *facility confirmation parameters* is defined for every facility. Any error is coded in the parameter *info*.

FACILITY_CONF	Command	**0x80**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Info	word	0: request accepted 0x2001: message not supported in current state 0x2002: incorrect Controller/PLCI/NCCI 0x2007: illegal message parameter coding 0x300B: facility not supported
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility confirmation parameter	struct	Facility-depending parameters

6.7.31 FACILITY_IND

Description

This message is used to indicate a facility dependent event originating from a controller or connections identified via *controller*/*PLCI*/*NCCI*, depending on the facility. The struct *facility indication parameter* is defined for every facility.

FACILITY_IND	Command	**0x80**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector (NOTE)	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility indication parameter	struct	Facility-depending parameters
NOTE: In case of facility selector **0** (Handset Support) this message may allocate a new PLCI (in case of off-hooking the handset) which shall be released afterwards by means of **DISCONNECT_IND**/**DISCONNECT_RESP**.		

6.7.32 FACILITY_RESP

Description

With this message the application confirms receipt of a facility indication message. The struct *facility response parameters* is defined for each facility.

FACILITY_RESP	Command	**0x80**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility response parameters	struct	Facility-depending parameters

6.7.33 INFO_REQ

Description

This message permits sending of protocol information for a the physical connection, e.g. overlap sending.

INFO_REQ	Command	**0x08**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller/PLCI	dword	See note
Called party number	struct	Called party number
Additional Info	struct	Additional information elements

NOTE: The first parameter identifies a physical connection (if a PLCI is given) or the addressed controller (if the PLCI field of parameter *Controller/PLCI* is zero). Depending on the parameter different messages shall be sent to the network.

6.7.34 INFO_CONF

Description

This message confirms acceptance of **INFO_REQ**. If in the corresponding **INFO_REQ** a controller is given as an addressing parameter, this connection is assigned a *PLCI* which serves as an identifier in further processing. Any error is coded in the parameter *info*.

INFO_CONF	Command	**0x08**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: transmission of information initiated 0x2001: message not supported in current state 0x2002: illegal Controller/PLCI 0x2003: out of PLCI 0x2007: illegal message parameter coding

6.7.35 INFO_IND

Description

This message indicates an event for a physical connection as expressed by an information element (*info element*) whose coding is described by the parameter *info number*. The connection is identified via the parameter *controller/PLCI*.

INFO_IND	Command	**0x08**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller/PLCI	dword	Physical Link Connection Identifier
Info number	word	Information element identifier
Info element	struct	Information element dependent structure

NOTE: An individual **INFO_IND** is displayed for each information element. To enable indication of events, the info mask parameter of the message **LISTEN_REQ** shall be used.
If the *PLCI* field in the address parameter is 0, the network has sent information not associated with a physical connection.
In case of getting information from the network which lead to other Profile B messages (e.g. receiving a RELEASE from the network which includes charging information) it is guaranteed that an application gets the **INFO_IND** first, followed by the corresponding Profile B message.

6.7.36 INFO_RESP

Description

With this message the application confirms the receipt of an **INFO_IND**.

INFO_RESP	Command	**0x08**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller/PLCI	dword	As in INFO_IND

6.7.37 LISTEN_REQ

Description

This message is used to activate signalling of incoming events from Profile B to the application. *Info mask* is used to define which signalling protocol events are indicated to the application. These events are normally associated with physical connections. *CIP mask* defines selection criteria based upon *Bearer Capability* and *High Layer Compatibility*, thus indicating which incoming calls are signalled to an application.

More than one application may listen to the same *CIP Values*. Every application listening to a matching value shall be informed about incoming calls. In case more than one application wants to accept the call, the first **CONNECT_RESP** received by Profile B as a reaction to the **CONNECT_IND** shall be accepted. Every other application shall get the message **DISCONNECT_IND** with a Parameter *reason* which indicates this situation.

This scenario is similar to the situation where more than one set of compatible ISDN equipment on an ISDN line attempts to accept an incoming call.

LISTEN_REQ	Command	**0x05**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller	dword	
Info mask	dword	Bit field, coding as follows: 0: cause 1: date/Time 2: display 3: user-user information 4: call progression 5: facility 6: charging 7 to 31: reserved
CIP Mask	dword	explained below
CIP Mask 2	dword	reserved for additional services
Calling party number	struct	Calling party number
Calling party subaddress	struct	Calling party subaddress

Explanation of *CIP Mask*:

Parameter	Type	Comment
CIP Mask	dword	Bit field, coding as follows: 0: any match 1: speech 2: unrestricted digital information 3: restricted digital information 4: 3.1 kHz audio 5: 7.0 kHz audio 6: video 7: packet mode 8: 56 kBit/s rate adaptation 9: unrestricted digital information with tones/announcements 10..15: reserved 16: telephony 17: fax group 2/3 18: fax group 4 class 1 19: Teletex service (basic and mixed), fax group 4 class 2 20: Teletex service (basic and processable) 21: Teletex service (basic) 22: Videotex 23: Telex 24: message handling systems according X.400 25: OSI applications according X.200 26: 7 kHz Telephony 27: Video Telephony F.721, first connection 28: Video Telephony F.721, second connection 29 to 31: reserved
NOTE: Clearing all bits in the *CIP mask* disables the signalling of incoming calls to the application. *Calling party number/subaddress* are only used for external ISDN equipment (handsets), which might need the *own* (local) address to handle *outgoing* calls.		

6.7.38 LISTEN_CONF

Description

This message confirms the acceptance of the LISTEN_REQ. Any errors are coded in the parameter *info*.

LISTEN_CONF	Command	**0x05**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller	dword	
Info	word	0: listen is active 0x2002: illegal controller 0x2005: out of LISTEN-Resources 0x2007: illegal message parameter coding

6.7.39 MANUFACTURER_REQ

Description

This message is used to transfer manufacturer specific information.

MANUFACTURER_REQ	Command	**0xFF**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

NOTE: This message should not be used, for it is a non compatible message. Applications which use this message may work only with one manufacturer of ISDN equipment.
A manufacturer shall choose *one* manufacturer specific ID for all of that Profile B implementations. This manufacturer specific ID shall be unique. A shortcut or nickname based on the manufacturer's initials might be a good choice.
The behaviour of Profile B is **not defined** after receiving any MANUFACTURER_REQ.

6.7.40 MANUFACTURER_CONF

Description

This message confirms the reception of a **MANUFACTURER_REQ**.

MANUFACTURER_CONF	Command	**0xFF**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

6.7.41 MANUFACTURER_IND

Description

This message is used to indicate manufacturer specific information to an application. Profile B shall not generate this message except it is requested by a **MANUFACTURER_REQ**.

MANUFACTURER_IND	Command	**0xFF**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data
NOTE: This message shall not be sent from Profile B without initial application request from an application by means of **MANUFACTURER_REQ**.		

6.7.42 MANUFACTURER_RESP

Description

With this message an application confirms receipt of a **MANUFACTURER_IND**.

MANUFACTURER_RESP	Command	**0xFF**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

6.7.43 RESET_B3_REQ

Description

With this message the specified logical connection is reset. The logical connection is identified by the parameter *NCCI*.

RESET_B3_REQ	Command	**0x87**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
The reaction to a **RESET_B3_REQ** depends on the selected layer 3 protocol. If ISO 8208 [3], T.90 [16], X.25 DCE or X.25 PLP in the D-channel was selected, the reset procedure is performed in accordance with the protocol recommendations. In case of a transparent layer 3, a reset procedure in layer 2 is initiated.
If a reset procedure is not defined for the protocol a **RESET_B3_REQ** causes the controller to generate a **RESET_B3_CONF** with info value **reset procedure not supported by current protocol** (0x300D). No further action is taken.
After successfully initiating a reset on a logical connection, an application is not allowed to transmit data until the resulting RESET_B3_IND (or DISCONNECT_B3_IND) message is received.
Loss of data may occur during reset procedure!

6.7.44 RESET_B3_CONF

Description

With this message the controller confirms the initiation of resetting a logical connection.

RESET_B3_CONF	Command	**0x87**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: reset initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x3008: NCPI not supported 0x300D: reset procedure not supported by current protocol

6.7.45 RESET_B3_IND

Description

With this message the resetting of a logical connection is indicated. The logical connection is identified by a *NCCI*.

RESET_B3_IND	Command	**0x87**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. In case of transparent layer 3 the re-establishment of layer 2 is indicated. This message may indicate a loss of data!		

6.7.46 RESET_B3_RESP

Description

With this message the application confirms the resetting of a logical connection.

RESET_B3_RESP	Command	**0x87**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier

6.7.47 SELECT_B_PROTOCOL_REQ

Description

This message allows an application to change the current protocol during the lifetime of a physical connection after receiving the message CONNECT_ACTIVE_IND. The support of this message is optional. If a particular Profile B implementation does not support this switching the info parameter of the corresponding **SELECT_B_PROTOCOL_CONF** shall be set to **message not supported in current state** (0x2001).

w

SELECT_B_PROTOCOL_REQ	Command	**0x41**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
B protocol	struct	Protocol definition

6.7.48 SELECT_B_PROTOCOL_CONF

Description

This message confirms the execution of switching the protocol stack for a physical connection. Any error shall be shown in *info*.

SELECT_B_PROTOCOL_CONF	Command	**0x41**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: protocol switch successful 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding 0x3001: B1 protocol not supported 0x3002: B2 protocol not supported 0x3003: B3 protocol not supported 0x3004: B1 protocol parameter not supported 0x3005: B2 protocol parameter not supported 0x3006: B3 protocol parameter not supported 0x3007: B protocol combination not supported

6.7.49 ALERT_REQ

Description

This message should be used by applications to indicate compatibility to an incoming call. It shall send an ALERT to the network to prevent the call from expiring (no user responding). If an application is able to accept the call immediately it shall not be necessary to use this message; the application can issue immediately a CONNECT_RESP to Profile B.

ALERT_REQ	Command	**0x01**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Additional info	struct	Additional info elements
NOTE: The parameter *Additional info* shall be coded as an empty structure if no additional information (e.g. user data) has to be transmitted.		

6.7.50 ALERT_CONF

Description

This message confirms the reception of an ALERT_REQ.

ALERT_CONF	Command	**0x01**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: alert initiated 0x0003: alert already sent by another application 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding
NOTE: Info 0x0003 shall be returned if another application already initiated the sending of an ALERT message to the network. In this case the parameter *Additional info* of the corresponding REQUEST has been ignored.		

See also: Description of broadcast mechanism in LISTEN_REQ

6.7.51 CONNECT_REQ

Description

This message initiates the set-up of a physical connection. An application may only offer the relevant parts of the parameters, i.e. *Controller*, *CIP Value, B protocol* and normally c*alled party number*. Every other structure can be empty (length of 0). In this case the default values as described in clause 6 shall be used.

CONNECT_REQ	Command	**0x02**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller	dword	
CIP Value	word	Compatibility Information Profile
Called party number	struct	Called party number
Calling party number	struct	Calling party number
Called party subaddress	struct	Called party subaddress
Calling party subaddress	struct	Calling party subaddress
B protocol	struct	B protocol to be used
BC	struct	Bearer Capability
LLC	struct	Low Layer Compatibility
HLC	struct	High Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: If an application offers *BC*, *LLC* and/or *HLC*, the parameter shall be used without checking the resulting combination.
The absence (i.e. coding as an empty structure) of *B protocol* shall result in the default protocol behaviour: ISO 7776 [4] (X.75) and window size 7. This is a recommended selection to get overall connectivity with the benefits of HDLC error recovery. Note that ISO 7776 [4] deals with a default maximum data length of 128 octets, whereas Profile B is able to handle up to at least 2 048 octets, depending on **CAPI_REGISTER** values of an application.

6.7.52 CONNECT_CONF

Description

This message confirms the initiation of a call set-up. This connection is assigned a *PLCI* which serves as an identifier in further processing. Errors are returned in the parameter *info*.

CONNECT_CONF	Command	**0x02**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: connect initiated 0x2002: illegal controller 0x2003: out of PLCI 0x2007: illegal message parameter coding 0x3001: B1 protocol not supported 0x3002: B2 protocol not supported 0x3003: B3 protocol not supported 0x3004: B1 protocol parameter not supported 0x3005: B2 protocol parameter not supported 0x3006: B3 protocol parameter not supported 0x3007: B protocol combination not supported 0x300A: CIP Value unknown

NOTE: The connection is in the set-up phase at this point in time. Subsequent successful switching is indicated by the message **CONNECT_ACTIVE_IND**.
If an application wants to identify the corresponding REQUEST to this message, it may use the message number mechanism described in subclause 6.2.

6.7.53 CONNECT_IND

Description

This message indicates an incoming call for a physical connection. For the incoming call a PLCI is assigned which is used to identify this connection in subsequent messages.

CONNECT_IND	Command	**0x02**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
CIP Value	word	Compatibility Information Profile
Called party number	struct	Called party number
Calling party number	struct	Calling party number
Called party subaddress	struct	Called party subaddress
Calling party subaddress	struct	Calling party subaddress
BC	struct	Bearer compatibility
LLC	struct	Low Layer Compatibility
HLC	struct	High Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: To activate the signalling of incoming calls, the message **LISTEN_REQ** shall be send to the controller.
Every information available from the network at this point shall be signalled to the application. Empty structs show the absence of this information.

6.7.54 CONNECT_RESP

Description

This message is used to accept or reject an incoming call on behalf of the application. The incoming call is identified via parameter *PLCI*. The parameter *reject* is used to accept, reject or ignore the call. In case of ignoring the call, other ISDN equipment connected on the same bus (basic access) shall have the chance to accept this call, whereas the rejection of this incoming call should terminate the call on the entire bus. For primary access, these parameter values of parameter *Reject* shall behave identically.

CONNECT_RESP	Command	**0x02**
	Subcommand	**0x83**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Reject	word	0: accept call 1: ignore call 2: reject call, normal call clearing 3: reject call, user busy 4: reject call, requestet circuit/channel not available 5: reject call, facility rejected 6: reject call, channel unacceptable 7: reject call, incompatible destination 8: reject call, destination out of order
B protocol	struct	B protocol to be used
Connected number	struct	Connected number
Connected subaddress	struct	Connected subaddress
LLC	struct	Low Layer Compatibility
Additional Info	struct	Additional information elements

NOTE: The parameter *LLC* can optionally be used for LLC negotiation if supported by the network.
Any unknown *reject* value shall be mapped to *normal call clearing*.
Any *reject* value other than *accept call* shall sent a DISCONNECT_IND to the application.
The absence (i.e. coding as an empty structure) of *B protocol* shall result in the default protocol behaviour: ISO 7776 [4] (X.75) and window size 7. This is a recommended selection to get overall connectivity with the benefits of HDLC error recovery. Note that ISO 7776 [4] deals with a default maximum data length of 128 octets, whereas Profile B is able to handle up to at least 2 048 octets, depending on **CAPI_REGISTER** values of an application.

6.7.55 CONNECT_ACTIVE_IND

Description

This message indicates the physical connection of a B-channel. The connection is identified by the parameter *PLCI*.

CONNECT_ACTIVE_IND	Command	**0x03**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Connected number	struct	Connected number
Connected subaddress	struct	Connected subaddress
LLC	struct	Low Layer Compatibility
NOTE: The parameter *connected number/subaddress* and *LLC* shall be filled in completely if this information is provided by the network. The absence of network information shall be indicated by empty structures.		

6.7.56 CONNECT_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_ACTIVE_IND**.

CONNECT_ACTIVE_RESP	Command	**0x03**
	Subcommand	**0x83**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier

6.7.57 CONNECT_B3_ACTIVE_IND

Description

This message indicates the logical connection of a B-channel. The connection is identified by the parameter *NCCI*. The parameter *NCPI* is used to transfer additional protocol dependent information.

CONNECT_B3_ACTIVE_IND	Command	**0x83**
	Subcommand	**0x82**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
After this message incoming data can be indicated to the application.
In case of protocol T.30 [14] and outgoing calls, this message does not imply the successful training between both facsimile stations. This is to enable an application to send data to Profile B without waiting for termination of training phase. If this training phase is not successful, corresponding indications shall be given by Profile B in the message DISCONNECT_B3_IND.

6.7.58 CONNECT_B3_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_B3_ACTIVE_IND**.

CONNECT_B3_ACTIVE_RESP	Command	**0x83**
	Subcommand	**0x83**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier

6.7.59 CONNECT_B3_REQ

Description

This message initiates the set-up of a logical connection. The physical connection is identified by the parameter *PLCI*. Additional protocol dependent information can be transferred with the parameter *NCPI*.

CONNECT_B3_REQ	Command	**0x82**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used.		

6.7.60 CONNECT_B3_CONF

Description

With this message the initiation of a logical connection set-up is confirmed. This connection is assigned a *NCCI*, which subsequently identifies this logical connection. Errors are supplied in the parameter *info*.

CONNECT_B3_CONF	Command	**0x82**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: connect initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2004: out of NCCI 0x3008: NCPI not supported
NOTE: The connection is in the set-up phase at this stage. The successful set-up shall be indicated by the message **CONNECT_B3_ACTIVE_IND**. If parameter *info* returns **0x0001**, the set-up of a logical connection is initiated, but parameter *NCPI* has been ignored. In that case the used layer 3 protocol does not support the usage of *NCPI* (e.g. the transparent mode of layer 3).		

6.7.61 CONNECT_B3_IND

Description

This message indicates an incoming call for a logical connection. For this incoming call a *NCCI* is assigned, which subsequently identifies the call. Additional protocol dependent information shall be transferred with parameter *NCPI* if available.

CONNECT_B3_IND	Command	**0x82**
	Subcommand	**0x82**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. The connection is in the set-up phase at this stage. The successful set-up shall be indicated by the message **CONNECT_B3_ACTIVE_IND**.		

6.7.62 CONNECT_B3_RESP

Description

With this message the application accepts or rejects an incoming logical call. The incoming call is identified via the parameter *NCCI*. The call can be accepted or rejected via the parameter *reject*. The parameter *NCPI* can be used to transfer additional protocol dependent information.

CONNECT_B3_RESP	Command	**0x82**
	Subcommand	**0x83**

Parameter	**Type**	**Comment**
NCCI	dword	Network Control Connection Identifier
Reject	word	0: accept call 2: reject call, normal call clearing
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. Any other value of *reject* shall result in rejecting the call.		

6.7.63 CONNECT_B3_T90_ACTIVE_IND

Description

This message indicates the switching from T.70 [15] to T.90 [16] within a logical connection of a B-channel. The connection is identified by the parameter *NCCI*. The parameter *NCPI* is used to transfer additional T.90 [16] information.

CONNECT_B3_T90_ACTIVE_IND	Command	**0x88**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: This message shall only be generated if the selected protocol is T.90NL with compatibility to T.70NL according to T.90 [16]Appendix II. In this case the initially used protool is T.70 [15]. This message indicates the negotiation and switching to T.90 [16].

6.7.64 CONNECT_B3_T90_ACTIVE_RESP

Description

With this message the application confirms the receipt of a **CONNECT_B3_T90_ACTIVE_IND**.

CONNECT_B3_T90_ACTIVE_RESP	Command	**0x88**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier

6.7.65 DATA_B3_REQ

Description

This message sends data within the logical connection identified by the *NCCI*. Data to be sent is referenced via the parameter *data*/*data length*. The data is not part of the message, a 32-bit pointer is used to transfer the address of the data area. The application issues a unique identifier for this data in the parameter *data handle*. On subsequent confirmation by a **DATA_B3_CONF** this handle is used. It is possible to set additional information, such as more data, delivery confirmation etc. via parameter *flags*. The flags are not supported by all protocols.

DATA_B3_REQ	Command	**0x86**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data	dword	Pointer to the data to be sent
Data length	word	Size of data area to be sent
Data handle	word	Referenced in DATA_B3_CONF
Flags	word	[0]: qualifier bit [1]: more data bit [2]: delivery confirmation bit [3]: expedited data [4] to [15]: reserved

NOTE: The data transfer does not support assembly or re-assembly of data.
An application shall not change or free the data area until the corresponding **DATA_B3_CONF** is received.
Flags are protocol dependent. If an application set reserved bits in parameter *Flags*, Profile B shall reject the **DATA_B3_REQ**. This is to allow future expansion of this parameter. If an application set bits in parameter *Flags*, which are not supported by the current protocol, Profile B shall accept the **DATA_B3_REQ** but shall return this information in the corresponding **DATA_B3_CONF**.

6.7.66 DATA_B3_CONF

Description

This message confirms the acceptance of a data package to be sent. The logical connection is identified by the parameter *NCCI*. The parameter *data handle* supplies the identifier used by the application in the associated **DATA_B3_REQ** as reference to the transferred data area. After receiving this message, the application can reuse the referenced data area.

DATA_B3_CONF	Command	**0x86**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data handle	word	Identifies the data area of corresponding DATA_B3_REQ
Info	word	0: data transmission initiated 0x0002: flags not supported by current protocol, flags ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x300A: flags not supported (reserved bits) 0x300C: data length not supported by current protocol

NOTE: Every **DATA_B3_REQ** shall result in a corresponding **DATA_B3_CONF** except in the following case: after transmitting the message **DISCONNECT_B3_IND** to an application, Profile B is not allowed to send any other message concerning this logical connection identified by the parameter NCCI. So in this case the application shall assure that resources or buffer management will be reset correctly.
If an application sets the delivery confirmation bit in the corresponding DATA_B3_REQ and the selected protocol supports this mechanism the confirmation shall be given to the application after the delivery of the sent packet is confirmed by the used protocol.
Seven unconfirmed DATA_B3_REQ messages shall be supported.

6.7.67 DATA_B3_IND

Description

This message indicates incoming data within a logical connection. The logical connection is identified via the *NCCI*. The length of the incoming data area is indicated via the parameter *data length*. The incoming data area can be referenced by the parameter *data*. The data is not part of the message, a 32-bit pointer is used to transfer the address of the data area. Profile B issues a handle to this data area via the parameter *data handle*. On subsequent confirmation by a **DATA_B3_RESP**, this handle shall also be supplied by the application. Additional information, such as more data, delivery confirmation etc. is supplied by parameter *flags*, if available.

DATA_B3_IND	Command	**0x86**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data	dword	Pointer to data received
Data length	word	Size of data area received
Data handle	word	handle to data area, referenced in DATA_B3_RESP
Flags	word	[0]: qualifier bit [1]: more-data bit [2]: delivery confirmation bit [3]: expedited data [4 to 14]: reserved [15]: framing error bit, data may be invalid (only with corresponding B2 protocol)

NOTE: The data transfer does not support re-assembly functions.
The data area which contains the data remains allocated until the corresponding **DATA_B3_RESP** is received. However, expedited data is only valid until the next **CAPI_GET_MESSAGE** is performed by the application.
In case of receiving **DATA_B3_IND** messages with reserved bits switched on in the flags parameter an application shall ignore the data area but process the message, i.e. send a **DATA_B3_RESP** to Profile B. This is to allow future expansion of the *flags* parameter.

6.7.68 DATA_B3_RESP

Description

With this message the application confirms acceptance of an incoming data package. The logical connection is identified by the parameter *NCCI*. The parameter *data handle* identifies the data handle used by Profile B in the corresponding **DATA_B3_IND** as the reference to the transferred data area.

DATA_B3_RESP	Command	**0x86**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Data handle	word	Data area reference in corresponding DATA_B3_IND

NOTE: This message frees the data buffer referenced by *Data handle* for reuse by Profile B.
Data throughput depends on an application's rapid response to DATA_B3_IND messages. Failure to do so shall trigger flow control on the line (for protocols supporting flow control such as ISO 7776 [4](X.75) or ISO 8208 [3](X.25)) and may cause loss of incoming data for protocols without flow control mechanism.

6.7.69 DISCONNECT_B3_REQ

Description

This message initiates the clearing of a logical connection identified via the parameter *NCCI*. The parameter *NCPI* can be used to transfer additional protocol dependent information.

DISCONNECT_B3_REQ	Command	**0x84**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
In case of facsimile group 3 (B protocol T.30 [14]) and speech (B1 protocol bit transparent, B2/B3 protocol transparent) data already given to transmission via **DATA_B3_REQ** shall be sent before disconnecting the logical connection.

6.7.70 DISCONNECT_B3_CONF

Description

With this message the initiation of clearing a logical connection is confirmed. Any errors are coded in the parameter *info*.

DISCONNECT_B3_CONF	Command	**0x84**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: disconnect initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x3008: NCPI not supported

6.7.71 DISCONNECT_B3_IND

Description

This message indicates the clearing of a logical connection identified via the parameter *NCCI*. The parameter *Reason_B3* indicates if this clearing is caused by wrong protocol behaviour. The parameter *NCPI* is used to indicate additional protocol dependent information if available.

DISCONNECT_B3_IND	Command	**0x84**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Reason_B3	word	0: clearing according to protocol 0x3301: protocol error layer 1 0x3302: protocol error layer 2 0x3303: protocol error layer 3 protocol dependent values are described in subclause 6.8
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the *NCPI* parameter depends on the protocol used.
After this message no other message concerning this *NCCI* shall be sent to the application. The application shall answer this message with **DISCONNECT_B3_RESP** to free the resources allocated to the *NCCI*.

6.7.72 DISCONNECT_B3_RESP

Description

With this message the application confirms the clearing of a logical connection.

DISCONNECT_B3_RESP	Command	**0x84**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NOTE: With this message resources allocated to the *NCCI* are released. If an application fails to send this message after receiving **DISCONNECT_B3_IND**, Profile B may reject subsequent **CONNECT_B3_REQ** with the info value **out of NCCI** (0x2004).		

6.7.73 DISCONNECT_REQ

Description

This message initiates the clearing of a physical connection, identified by the parameter *PLCI*.

DISCONNECT_REQ	Command	**0x04**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Additional Info	struct	Additional information elements
NOTE: Existing logical connections shall be cleared by Profile B using the message DISCONNECT_B3_IND containing the cause **protocol error layer 1** (0x3301) before clearing the physical connection.		

6.7.74 DISCONNECT_CONF

Description

This message confirms the initiation of clearing a physical connection. Any errors are coded in the parameter *info*.

DISCONNECT_CONF	Command	**0x04**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: disconnect initiated 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding

6.7.75 DISCONNECT_IND

Description

This message indicates the clearing of the physical channel identified via the parameter *PLCI*. The parameter *reason* indicates the network delivered cause or if this clearing is caused by wrong protocol behaviour

DISCONNECT_IND	Command	**0x04**
	Subcommand	**0x82**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Reason	word	0: no cause available 0x3301: protocol error layer 1 0x3302: protocol error layer 2 0x3303: protocol error layer 3 0x3304: another application got that call 0x34xx: disconnect cause from the network according to Q.931 [8]/ETS 300 102-1 [2]. In the field 'xx' the cause value received within a cause information element (octet 4) from the network is indicated.
NOTE: After this message no other message concerning this *PLCI* shall be sent to the application. The application shall answer this message with **DISCONNECT_RESP** to free the resources allocated to the *PLCI*.		

6.7.76 DISCONNECT_RESP

Description

With this message the application confirms the clearing of the physical channel.

DISCONNECT_RESP	Command	**0x04**
	Subcommand	**0x83**

Parameter	**Type**	**Comment**
PLCI	dword	Physical Link Connection Identifier
NOTE: With this message the *PLCI* is released. If an application fails to send this message after receiving **DISCONNECT_IND** resources bound to this *PLCI* shall not be freed. This may lead to Profile B resource problems (indicated by info value **out of PLCI),** affecting other applications too.		

6.7.77 FACILITY_REQ

Description

This message is used to handle optional facilities on the *controller* or facilities related on connections identified by *PLCI* or *NCCI*. The struct *facility request parameters* is defined for each facility. At the moment facilities Handset Support and DTMF are defined. Handset Support is used to support external ISDN equipment, DTMF is used in the Public Switched Telephone Network (PSTN) to select and control several provided services (e.g. automatic answering service).

Handset Support as well as DTMF support are optional Profile B features. In case Profile B does not support these facilities, an appropriate information value is returned in the **FACILITY_CONF**.

DTMF cannot be used with all B protocols. Normally, it is used with 64 kbit/s bit speech and T.30 [14] audio.

FACILITY_REQ	Command	**0x80**
	Subcommand	**0x80**

Parameter	**Type**	**Comment**
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility request parameter	struct	Facility depending parameters

6.7.78 FACILITY_CONF

Description

This message confirms the acceptance of the **FACILITY_REQ**. The event is identified by *Controller/PLCI/NCCI*, depending on the facility. The struct *facility confirmation parameters* is defined for every facility. Any error is coded in the parameter *info*.

FACILITY_CONF	Command	**0x80**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Info	word	0: request accepted 0x2001: message not supported in current state 0x2002: incorrect Controller/PLCI/NCCI 0x2007: illegal message parameter coding 0x300B: facility not supported
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility confirmation parameter	struct	Facility-depending parameters

6.7.79 FACILITY_IND

Description

This message is used to indicate a facility dependent event originating from a controller or connections identified via *controller*/*PLCI*/*NCCI*, depending on the facility. The struct *facility indication parameter* is defined for every facility.

FACILITY_IND	Command	**0x80**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector (NOTE)	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility indication parameter	struct	Facility-depending parameters
NOTE: In case of facility selector **0** (Handset Support) this message may allocate a new PLCI (in case of off-hooking the handset) which shall be released afterwards by means of **DISCONNECT_IND**/**DISCONNECT_RESP**.		

6.7.80 FACILITY_RESP

Description

With this message the application confirms receipt of a facility indication message. The struct *facility response parameters* is defined for each facility.

FACILITY_RESP	Command	**0x80**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller/PLCI/NCCI	dword	Depending on the facility selector
Facility selector	word	0: Handset Support 1: DTMF 2 to n: reserved
Facility response parameters	struct	Facility-depending parameters

6.7.81 INFO_REQ

Description

This message permits sending of protocol information for a the physical connection, e.g. overlap sending.

INFO_REQ	Command	**0x08**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller/PLCI	dword	See note
Called party number	struct	Called party number
Additional Info	struct	Additional information elements

NOTE: The first parameter identifies a physical connection (if a PLCI is given) or the addressed controller (if the PLCI field of parameter *Controller/PLCI* is zero). Depending on the parameter different messages shall be sent to the network.

6.7.82 INFO_CONF

Description

This message confirms acceptance of **INFO_REQ**. If in the corresponding **INFO_REQ** a controller is given as an addressing parameter, this connection is assigned a *PLCI* which serves as an identifier in further processing. Any error is coded in the parameter *info*.

INFO_CONF	Command	**0x08**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: transmission of information initiated 0x2001: message not supported in current state 0x2002: illegal Controller/PLCI 0x2003: out of PLCI 0x2007: illegal message parameter coding

6.7.83 INFO_IND

Description

This message indicates an event for a physical connection as expressed by an information element (*info element*) whose coding is described by the parameter *info number*. The connection is identified via the parameter *controller/PLCI*.

INFO_IND	Command	**0x08**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller/PLCI	dword	Physical Link Connection Identifier
Info number	word	Information element identifier
Info element	struct	Information element dependent structure

NOTE: An individual **INFO_IND** is displayed for each information element. To enable indication of events, the info mask parameter of the message **LISTEN_REQ** shall be used.
If the *PLCI* field in the address parameter is 0, the network has sent information not associated with a physical connection.
In case of getting information from the network which lead to other Profile B messages (e.g. receiving a RELEASE from the network which includes charging information) it is guaranteed that an application gets the **INFO_IND** first, followed by the corresponding Profile B message.

6.7.84 INFO_RESP

Description

With this message the application confirms the receipt of an **INFO_IND**.

INFO_RESP	Command	**0x08**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller/PLCI	dword	As in INFO_IND

6.7.85 LISTEN_REQ

Description

This message is used to activate signalling of incoming events from Profile B to the application. *Info mask* is used to define which signalling protocol events are indicated to the application. These events are normally associated with physical connections. *CIP mask* defines selection criteria based upon *Bearer Capability* and *High Layer Compatibility*, thus indicating which incoming calls are signalled to an application.

More than one application may listen to the same *CIP Values*. Every application listening to a matching value shall be informed about incoming calls. In case more than one application wants to accept the call, the first **CONNECT_RESP** received by Profile B as a reaction to the **CONNECT_IND** shall be accepted. Every other application shall get the message **DISCONNECT_IND** with a Parameter *reason* which indicates this situation.

This scenario is similar to the situation where more than one set of compatible ISDN equipment on an ISDN line attempts to accept an incoming call.

LISTEN_REQ	Command	**0x05**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller	dword	
Info mask	dword	Bit field, coding as follows: 0: cause 1: date/Time 2: display 3: user-user information 4: call progression 5: facility 6: charging 7 to 31: reserved
CIP Mask	dword	explained below
CIP Mask 2	dword	reserved for additional services
Calling party number	struct	Calling party number
Calling party subaddress	struct	Calling party subaddress

Explanation of *CIP Mask*:

Parameter	Type	Comment
CIP Mask	dword	Bit field, coding as follows: 0: any match 1: speech 2: unrestricted digital information 3: restricted digital information 4: 3.1 kHz audio 5: 7.0 kHz audio 6: video 7: packet mode 8: 56 kBit/s rate adaptation 9: unrestricted digital information with tones/announcements 10..15: reserved 16: telephony 17: fax group 2/3 18: fax group 4 class 1 19: Teletex service (basic and mixed), fax group 4 class 2 20: Teletex service (basic and processable) 21: Teletex service (basic) 22: Videotex 23: Telex 24: message handling systems according X.400 25: OSI applications according X.200 26: 7 kHz Telephony 27: Video Telephony F.721, first connection 28: Video Telephony F.721, second connection 29 to 31: reserved
NOTE: Clearing all bits in the *CIP mask* disables the signalling of incoming calls to the application. *Calling party number/subaddress* are only used for external ISDN equipment (handsets), which might need the *own* (local) address to handle *outgoing* calls.		

6.7.86 LISTEN_CONF

Description

This message confirms the acceptance of the LISTEN_REQ. Any errors are coded in the parameter *info*.

LISTEN_CONF	Command	**0x05**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller	dword	
Info	word	0: listen is active 0x2002: illegal controller 0x2005: out of LISTEN-Resources 0x2007: illegal message parameter coding

6.7.87 MANUFACTURER_REQ

Description

This message is used to transfer manufacturer specific information.

MANUFACTURER_REQ	Command	**0xFF**
	Subcommand	**0x80**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

NOTE: This message should not be used, for it is a non compatible message. Applications which use this message may work only with one manufacturer of ISDN equipment.
A manufacturer shall choose *one* manufacturer specific ID for all of that Profile B implementations. This manufacturer specific ID shall be unique. A shortcut or nickname based on the manufacturer's initials might be a good choice.
The behaviour of Profile B is **not defined** after receiving any MANUFACTURER_REQ.

6.7.88 MANUFACTURER_CONF

Description

This message confirms the reception of a **MANUFACTURER_REQ**.

MANUFACTURER_CONF	Command	**0xFF**
	Subcommand	**0x81**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

6.7.89 MANUFACTURER_IND

Description

This message is used to indicate manufacturer specific information to an application. Profile B shall not generate this message except it is requested by a **MANUFACTURER_REQ**.

MANUFACTURER_IND	Command	**0xFF**
	Subcommand	**0x82**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data
NOTE: This message shall not be sent from Profile B without initial application request from an application by means of **MANUFACTURER_REQ**.		

6.7.90 MANUFACTURER_RESP

Description

With this message an application confirms receipt of a **MANUFACTURER_IND**.

MANUFACTURER_RESP	Command	**0xFF**
	Subcommand	**0x83**

Parameter	Type	Comment
Controller	dword	
Manu ID	dword	Manufacturer specific ID (should be unique)
Manufacturer specific		Manufacturer specific data

6.7.91 RESET_B3_REQ

Description

With this message the specified logical connection is reset. The logical connection is identified by the parameter *NCCI*.

RESET_B3_REQ	Command	**0x87**
	Subcommand	**0x80**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information

NOTE: The meaning of the parameter *NCPI* depends on the protocol used.
The reaction to a **RESET_B3_REQ** depends on the selected layer 3 protocol. If ISO 8208 [3], T.90 [16], X.25 DCE or X.25 PLP in the D-channel was selected, the reset procedure is performed in accordance with the protocol recommendations. In case of a transparent layer 3, a reset procedure in layer 2 is initiated.
If a reset procedure is not defined for the protocol a **RESET_B3_REQ** causes the controller to generate a **RESET_B3_CONF** with info value **reset procedure not supported by current protocol** (0x300D). No further action is taken.
After successfully initiating a reset on a logical connection, an application is not allowed to transmit data until the resulting RESET_B3_IND (or DISCONNECT_B3_IND) message is received.
Loss of data may occur during reset procedure!

6.7.92 RESET_B3_CONF

Description

With this message the controller confirms the initiation of resetting a logical connection.

RESET_B3_CONF	Command	**0x87**
	Subcommand	**0x81**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
Info	word	0: reset initiated 0x0001: NCPI not supported by current protocol, NCPI ignored 0x2001: message not supported in current state 0x2002: illegal NCCI 0x2007: illegal message parameter coding 0x3008: NCPI not supported 0x300D: reset procedure not supported by current protocol

6.7.93 RESET_B3_IND

Description

With this message the resetting of a logical connection is indicated. The logical connection is identified by a *NCCI*.

RESET_B3_IND	Command	**0x87**
	Subcommand	**0x82**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier
NCPI	struct	Network Control Protocol Information
NOTE: The meaning of the parameter *NCPI* depends on the protocol used. In case of transparent layer 3 the re-establishment of layer 2 is indicated. This message may indicate a loss of data!		

6.7.94 RESET_B3_RESP

Description

With this message the application confirms the resetting of a logical connection.

RESET_B3_RESP	Command	**0x87**
	Subcommand	**0x83**

Parameter	Type	Comment
NCCI	dword	Network Control Connection Identifier

6.7.95 SELECT_B_PROTOCOL_REQ

Description

This message allows an application to change the current protocol during the lifetime of a physical connection after receiving the message CONNECT_ACTIVE_IND. The support of this message is optional. If a particular Profile B implementation does not support this switching the info parameter of the corresponding **SELECT_B_PROTOCOL_CONF** shall be set to **message not supported in current state** (0x2001).

SELECT_B_PROTOCOL_REQ	Command	**0x41**
	Subcommand	**0x80**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
B protocol	struct	Protocol definition

6.7.96 SELECT_B_PROTOCOL_CONF

Description

This message confirms the execution of switching the protocol stack for a physical connection. Any error shall be shown in *info*.

SELECT_B_PROTOCOL_CONF	Command	**0x41**
	Subcommand	**0x81**

Parameter	Type	Comment
PLCI	dword	Physical Link Connection Identifier
Info	word	0: protocol switch successful 0x2001: message not supported in current state 0x2002: illegal PLCI 0x2007: illegal message parameter coding 0x3001: B1 protocol not supported 0x3002: B2 protocol not supported 0x3003: B3 protocol not supported 0x3004: B1 protocol parameter not supported 0x3005: B2 protocol parameter not supported 0x3006: B3 protocol parameter not supported 0x3007: B protocol combination not supported

6.8 Parameter descriptions

This subclause describes the parameters used in Profile B messages. Each parameter is listed with its type, possible values and reference to the messages in which the parameter appears.

Some parameter values are defined according to ETS 300 102-1 [2] or Q.931 [8]. In that case there is no private Profile B coding for these parameters. These parameters are coded as Profile B structures starting with a length octet and the remainder of the parameter being coded as defined in ETS 300 102-1 [2]/Q.931 [8] from octet three onwards. References to the contents of a structure in this clause always use index 0 to identify the first octet of information, i.e. the octet following the length octet.

Parameters may not be omitted, instead an empty structure shall be used. An empty structure shall be coded as a single octet containing a value of 0.

Default values as described in the following section shall be implemented in Profile B. They need not be valid for external ISDN equipment; in that case the external equipment defines the default values for its usage.

Parameters may again contain parameters which are referred to as "sub parameters".

6.8.1 Additional Info

Additional Info (struct)

The purpose of the parameter *additional info* is to exchange signalling protocol specific information of the network. Depending on the signalling protocol only relevant elements of this structure shall be used (e.g. the B-channel information has to be ignored in the message DISCONNECT_REQ).

The parameter has the following structure:

struct **B-channel information**
struct **Keypad facility (coded according to ETS 300 102-1 [2]/Q.931 [8])**
struct **User data (coded according to ETS 300 102-1 [2]/Q.931 [8])**

struct	**Facility data array, which is used to transfer additional parameters coded according to ETS 300 102-1 [2]/Q.931 [8] starting from octet 1. This field is used to transport one or more complete facility data information elements.**

This information element appears in:

ALERT_REQ

CONNECT_REQ

CONNECT_IND

CONNECT_RESP

DISCONNECT_REQ

INFO_REQ

6.8.2 B-channel Information

B-channel Information (struct)

The purpose of the sub parameter *B-channel information* is to choose between B-channel data exchange, D-channel data exchange or pure user-user data exchange. If this struct is empty the default value is assumed.

This sub parameter is coded as a structure, to give an easy way of extending its contents in future changes. At the moment, it is coded as a structure of two bytes length and has one element:

word Channel:

0: use B-channel (default value)
1: use D-channel
2: use neither B-channel nor D-channel

This sub parameter appears in parameter:

Additional information

6.8.3 B Protocol

B Protocol (struct)

The purpose of the parameter *B protocol* is to select and configure the B-channel protocols. There is a protocol identifier and configuration information for each layer. If this struct is empty the default value is assumed.

The parameter has the following structure:

word	**B1 protocol: Physical layer and framing**
word	**B2 protocol: Data link layer**
word	**B3 protocol: Network layer**
struct	**B1 configuration: Physical layer and framing parameter**
struct	**B2 configuration: Data link layer parameter**
struct	**B3 configuration: Network layer parameter**

This information element appears in:

CONNECT_REQ

CONNECT_RESP

SELECT_B_PROTOCOL_REQ

6.8.4 B1 Protocol

B1 Protocol (word)

The purpose of the sub parameter *B1 protocol* is to specify the physical layer and framing used for this connection.

The following values are defined:

0: **64 kBit/s with HDLC framing. This is the default B1 protocol.**
1: **64 kBit/s bit transparent operation with byte framing from the network**
2: **V.110 [17] asynchronous operation with start/stop byte framing**
3: **V.110 [17] synchronous operation with HDLC framing**
4: **T.30 [14] modem for fax group 3**
5: **64 kBit/s inverted with HDLC framing.**
6: **56 kBit/s bit transparent operation with byte framing from the network**

This sub parameter appears in parameter:

B protocol

6.8.5 B2 Protocol

B2 Protocol (word)

The purpose of the sub parameter *B2 protocol* is to specify the data link layer used for this connection.

The following values are defined:

0: **ISO 7776 [4] (X.75 SLP) This is the default B2 protocol.**
1: **Transparent**
2: **SDLC [12]**
3: **LAPD according Q.921 [13] for D-channel X.25**
4: **T.30 [14] for facsimile group 3**
5: **Point-to-Point Protocol (PPP [10] [11])**
6: **Transparent (ignoring framing errors of B1 protocol)**

This sub parameter appears in parameter:

B protocol

6.8.6 B3 Protocol

B3 Protocol (word)

The purpose of the sub parameter *B3 protocol* is to specify the network layer used for this connection.

The following values are defined:

0:	**Transparent. This is the default B3 protocol**
1:	**T.90NL with compatibility to T.70NL according to T.90 [16] Appendix II.**
2:	**ISO 8208 [3] (X.25 DTE-DTE)**
3:	**X.25 DCE**
4:	**T.30 [14] for facsimile group 3**

This sub parameter appears in parameter:

B protocol

6.8.7 B1 Configuration

B1 Configuration (struct)

The purpose of the sub parameter *B1 configuration* is to offer additional configuration information for the B1 protocol. The parameter has the following structure:

word	Rate	This parameter has different meaning and default values depending on the selected B1 protocol: **### B1 protocol 0:** not applicable **### B1 protocol 1:** not applicable **### B1 protocol 2:** the maximum bit rate, coded as unsigned integer value. Default: adaptive **### B1 protocol 3:** the maximum bit rate, coded as unsigned integer value. Default: 56 kBit **### B1 protocol 4:** the maximum bit rate, coded as unsigned integer value. Default: adaptive **### B1 protocol 5:** not applicable **### B1 protocol 6:** not applicable
word	Bits per character/ Transmit Level	This parameter has different meaning and default values depending on the selected B1 protocol: **### B1 protocol 0:** not applicable **### B1 protocol 1:** not applicable **### B1 protocol 2:** bits per character, coded as unsigned integer value. Default: 8 **### B1 protocol 3:** not applicable **### B1 protocol 4:** the level is coded as signed integer specifying dB's. If this parameter or its value is not supported by the ISDN controller, it is ignored. **### B1 protocol 5:** not applicable **### B1 protocol 6:** not applicable
word	parity	This parameter has different meaning and default values depending on the selected B1 protocol: **### B1 protocol 0:** not applicable **### B1 protocol 1:** not applicable **### B1 protocol 2:** Parity: 0: none, 1: odd, 2: even. Default: no parity **### B1 protocol 3:** not applicable **### B1 protocol 4:** not applicable **### B1 protocol 5:** not applicable **### B1 protocol 6:** not applicable
word	stop bits	This parameter has different meaning and default values depending on the selected B1 protocol: **### B1 protocol 0:** not applicable **### B1 protocol 1:** not applicable **### B1 protocol 2:** stop bits: 0: 1 stop bit, 1: 2 stop bit. Default: 1 stop bit **### B1 protocol 3:** not applicable **### B1 protocol 4:** not applicable **### B1 protocol 5:** not applicable **### B1 protocol 6:** not applicable

This sub parameter appears in parameter:

B protocol

6.8.8 B2 Configuration

B2 Configuration (struct)

The purpose of the sub parameter *B2 configuration* is to offer additional configuration information for B2 protocol. It is only used for B2 protocols 0, 2 and 3. The parameter has the following structure:

byte	Address A	This parameter has different meaning and default values depending on the selected B2 protocol: ### **B2 protocol 0:** link Address A, default is 0x03 ### **B2 protocol 2:** link Address, default is 0xC1 ### **B2 protocol 3:** bit 0: ´0´ - automatic TEI assignment procedure shall be used. ´1´ - the TEI value shall be used as fixed TEI. In this case Bit 7 - Bit 1: TEI value
byte	Address B	This parameter has different meaning and default values depending on the selected B2 protocol: ### **B2 protocol 0:** link Address B, default is 0x01 ### **B2 protocol 2:** not applicable ### **B2 protocol 3:** not applicable
byte	Modulo Mode	Mode of operation: ### 8 - normal operation (Default) ### 128 - extended operation
byte	Window Size	Window size, default is 7.
struct	XID	This parameter has different meaning and default values depending on the selected B2 protocol: ### **B2 protocol 0:** not applicable ### **B2 protocol 2:** this is the content of the XID response which is sent when a XID command is received. ### **B2 protocol 3:** not applicable

This sub parameter appears in parameter:

B protocol

6.8.9 B3 Configuration

B3 Configuration (struct)

The purpose of the sub parameter *B3 configuration* is to offer additional configuration information for B3 protocol. Different structures of this parameter are defined, depending on the B3 protocol:

For B3 protocols 0 (transparent) this parameter does not apply (coded as an empty structure).

For B3 protocols 1, 2 and 3 (T.90NL, ISO 8208 [3], X.25 DCE) the following structure is defined:

word	LIC	Lowest incoming channel, default is 0
word	HIC	Highest incoming channel, default is 0
word	LTC	Lowest two-way channel, default is 1
word	HTC	Highest two-way channel, default is 1
word	LOC	Lowest outgoing channel, default is 0
word	HOC	Highest outgoing channel, default is 0
word	Modulo Mode	Mode of operation: ### 8 - normal operation (default) ### 128 - extended operation
word	Window Size	Used to configure non-standard defaults for the transmit and receive window size, default is 2

For B3 protocol 4 (Facsimile G3) the following structure is used:

word	resolution	0: standard 1: high
word	format	0: SFF (Default, description in annex B) 1: Plain FAX Format (modified Huffman coding) 2: PCX 3: DCX 4: TIFF 5: ASCII 6: Extended ANSI 7: Binary-File transfer
struct	station id	ID of the calling station. Coded in ASCII
struct	head line	Headline sent on each fax page. Coded in ASCII

This sub parameter appears in parameter:

B protocol

6.8.10 BC

BC (struct)

The purpose of the parameter *Bearer Capability (BC)* information element is to indicate a requested CCITT Recommendation I.231 bearer service to be provided by the network. It contains only information which may be used by the network. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

This information element appears in:

CONNECT_IND

CONNECT_REQ

6.8.11 Called Party Number

Called Party Number (struct)

The purpose of the parameter *called party number* information element is to identify the called party of a call. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

Byte 0	Type of number and numbering plan identification (byte 3 of the *called party number* information element, see ETS 300 102 [2]). At the calling side the value supplied by the application shall be transmitted over the network, **0x80** is the suggested default value. At the called side the value received from the network shall be passed to the application.
Bytes 1..n	Number digits of the *called party number* information element.

This information element appears in:

CONNECT_IND

CONNECT_REQ

6.8.12 Called Party Subaddress

Called Party Subaddress (struct)

The purpose of the parameter c*alled party subaddress* is to identify the subaddress of the called party of a call. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

Byte 0	Type of subaddress At the calling side the value supplied by application shall be transmitted over the network, **0x80** is the suggested default value (NSAP according X.213 [6]). In this case, the first subaddress information octet should have the value **0x50**. At the called side, the value received from the network shall be passed to the application.
Bytes 1..n	Contents of the called party subaddress information element.

This information element appears in:

CONNECT_REQ

CONNECT_IND

6.8.13 Calling Party Number

Calling Party Number (struct)

The purpose of the parameter c*alling party number* information element is to identify the origin of a call. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

Byte 0	Type of number and numbering plan identification (byte 3 of the *calling party number* information element, see ETS 300 102 [2]). At the calling side the value supplied by the application shall be transmitted over the network, **0x00** is the suggested default value. At the called interface the value received from the network shall be passed to the application. The extension bit shall always be cleared.
Byte 1	Presentation and screening indicator (byte 3a of the c*alling party number* information element). This byte may be used to allow or suppress the presentation of the caller's number in an incoming call. At the originating interface the value supplied by the application shall be transmitted over the network, **0x80** is the suggested default value. With this default value the presentation of the callers number is allowed. **0xA0** shall suppress the presentation of the calling number, if the network supports this mechanism. At the called interface the value received from the network shall be passed to the application. If this byte was not transmitted from the network, the controller inserts the valid default value **0x80** (user provided, not screened).
Bytes 2..n	Number digits of the *calling party number* information element.

This information element appears in:

CONNECT_REQ

CONNECT_IND

LISTEN_REQ

6.8.14 Calling Party Subaddress

Calling Party Subaddress (struct)

The purpose of the parameter c*alling party subaddress* information element is to identify a subaddress associated with the origin of a call. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

Byte 0	Type of subaddress At the calling side the value supplied by application shall be transmitted over the network, **0x80** is the suggested default value (NSAP according X.213 [6]). In this case, the first subaddress information octet should have the value **0x50**. At the called side, the value received from the network shall be passed to the application.
Bytes 1..n	Contents of the calling party subaddress information element.

This information element appears in:

CONNECT_IND

CONNECT_REQ

LISTEN_REQ

6.8.15 CIP Value

CIP Value (word)

The purpose of parameter *CIP Value* is to identify a complete profile of compatibility information (*Bearer Capability*, *Low Layer Compatibility* and *High Layer Compatibility*). With this parameter standard applications are not required to do complex coding and decoding of the above mentioned information elements.

Some of the *CIP* values only define a *Bearer Capability* (*CIP* 1 to 9) and some values define a combination of *Bearer Capability* and *High Layer Compatibility* (*CIP* 16 to 28). A *Low Layer Compatibility* information element is not defined with the *CIP*. The *Low Layer Compatibility* information element may be provided by the application if necessary.

The following *CIP* values are defined:

CIP value	Service	Relation to BC/HLC
0		no predefined profile
1	Speech	Bearer capability: coding standard: CCITT information transfer capability: speech transfer mode: circuit mode information transfer rate: 64 kBit/s user information layer 1 protocol: G.711 Coding of BC: <0x04, 0x03, 0x80, 0x90, 0xA3> or <0x04, 0x03, 0x80, 0x90, 0xA2>(see note)
2	unrestricted digital information	Bearer capability: coding standard: CCITT information transfer capability: unrestricted digital information transfer mode: circuit mode information transfer rate: 64 kBit/s Coding of BC: <0x04, 0x02, 0x88, 0x90>
3	restricted digital information	Bearer capability: coding standard: CCITT information transfer capability: restricted digital information transfer mode: circuit mode information transfer rate: 64 kBit/s Coding of BC: <0x04, 0x02, 0x89, 0x90>
4	3.1 kHz audio	Bearer capability: coding standard: CCITT information transfer capability: 3,1 kHz audio transfer mode: circuit mode information transfer rate: 64 kBit/s user information layer 1 protocol: G.711 Coding of BC: <0x04, 0x03, 0x90, 0x90, 0xA3> or <0x04, 0x03, 0x80, 0x90, 0xA2>(see note)
5	7 kHz audio	Bearer capability: coding standard: CCITT information transfer capability: unrestricted digital information with tones/announcements (this codepoint was formally labelled '7 kHz audio') transfer mode: circuit mode information transfer rate: 64 kBit/s Coding of BC: <0x04, 0x02, 0x91, 0x90>
6	video	Bearer capability: coding standard: CCITT information transfer capability: video transfer mode: circuit mode information transfer rate: 64 kBit/s Coding of BC: <0x04, 0x02, 0x98, 0x90>

(continued)

CIP value	Service	Relation to BC/HLC
7	packet mode	Bearer capability: coding standard: CCITT information transfer capability: unrestricted digital information transfer mode: packet mode information transfer rate: packet mode layer 2 protocol: X.25 layer 2 layer 3 protocol: X.25 layer 3 Coding of BC: <0x04, 0x04, 0x88, 0xC0, 0xC6, 0xE6>
8	56 kBit/s rate adaptation	Bearer capability: coding standard: CCITT information transfer capability: unrestricted digital information transfer mode: circuit mode layer 1 protocol: CCITT standardised rate adaptation V.110 [17]/X.30 [18] information transfer rate: packet mode rate: 56 kBit/s Coding of BC: <0x04, 0x04, 0x88, 0x90, 0x21, 0x8F>
9	unrestricted digital information with tones/announcements	Bearer capability: coding standard: CCITT information transfer capability: unrestricted digital information with tones/announcements (this codepoint was formally labelled '7 kHz audio') transfer mode: circuit mode information transfer rate: 64 kBit/s layer 1 protocol: H.221, H.242 Coding of BC: <0x05, 0x02, 0x91, 0x90, 0xA5>
10..15	reserved	
16	Telephony	Bearer Capability according to CIP 1. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Telephony Coding of HLC: <0x7D, 0x02, 0x91, 0x81>

(continued)

CIP value	Service	Relation to BC/HLC
17	Facsimile Group 2/3	Bearer Capability according to CIP 4. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Facsimile Group 2/3 Coding of HLC: <0x7D, 0x02, 0x91, 0x84>
18	Facsimile Group 4 Class 1	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Facsimile Group 4 Class 1 Coding of HLC: <0x7D, 0x02, 0x91, 0xA1>
19	Teletex service basic and mixed mode and facsimile service Group 4 Classes II and III	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification. Teletex service and facsimile service Group 4 Coding of HLC: <0x7D, 0x02, 0x91, 0xA4>
20	Teletex service basic and processable mode	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification. Teletex service basic and processable mode Coding of HLC: <0x7D, 0x02, 0x91, 0xA8>
21	Teletex service basic mode	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification. Teletex service basic mode Coding of HLC: <0x7D, 0x02, 0x91, 0xB1>

(continued)

CIP value	Service	Relation to BC/HLC
22	International inter working for Videotex	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification. International inter working for Videotex Coding of HLC: <0x7D, 0x02, 0x91, 0xB2>
23	Telex	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Telex Coding of HLC: <0x7D, 0x02, 0x91, 0xB5>
24	Message Handling Systems according to X.400 [20]	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Message Handling Systems according X.400 [20] Coding of HLC: <0x7D, 0x02, 0x91, 0xB8>
25	OSI application according to X.200 [19]	Bearer Capability according to CIP 2. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: OSI application according X.200 [19] Coding of HLC: <0x7D, 0x02, 0x91, 0xC1>
26	7 kHz Telephony	Bearer Capability according to CIP 9. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Telephony Coding of HLC: <0x7D, 0x02, 0x91, 0x81>

(continued)

CIP value	Service	Relation to BC/HLC
27	Video Telephony, first connection	Bearer Capability according to CIP 9. High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Video telephony (Rec. F.721) Extended high layer characteristics identification: Capability set of initial channel of H.221 Coding of HLC: <0x7D, 0x03, 0x91, 0xE0, 0x01>
28	Video Telephony, second connection	Bearer Capability according to CIP 2 High Layer Compatibility: coding standard: CCITT interpretation: First characteristics identification is to be used Presentation: High layer protocol profile High layer characteristics identification: Video telephony (Rec F.721) Extended high layer characteristics identification: Capability set of subsequent channel of H.221 Coding of HLC: <0x7D, 0x03, 0x91, 0xE0, 0x02>
NOTE:	This coding applies to ISDN with a default of A-Law coding for speech/audio. For ISDN with a default of ### Law coding the corresponding values shall be used.	

This information element appears in:

CONNECT_REQ
CONNECT_IND

6.8.16 CIP mask

CIP mask (dword)

The purpose of the parameter *CIP mask* is to select basic classes of incoming calls. The bit position within this mask identifies the related CIP value. When an incoming call is received, Profile B tries to match this incoming call to the defined CIP values (more than one value may match). A CONNECT_IND message is sent to the application when the bit position within the *CIP mask* of any matching CIP value is set to ´1´. The CIP value in the CONNECT_IND message is set to the highest matching CIP value.

The following rules are defined to find matching CIPs:

1) CIP values which define a Bearer Capability only (CIP 1 to CIP 9) shall generate a match with any incoming call which includes a Bearer Capability with the same information. Additional information included in the Bearer Capability information element shall be ignored. The match shall be generated regardless of any Low Layer Compatibility or High Layer Compatibility received.

2) CIP values which define a Bearer Capability and a High Layer Compatibility (CIP 16 to CIP 28) shall generate a match with any incoming call which includes a Bearer Capability and a High Layer Compatibility with the same identical information. The match shall be generated regardless of any Low Layer Compatibility received.

Bit 0 in the *CIP mask* has a special meaning. When no other matching bit is set in the *CIP mask* but the Bit 0, a CONNECT_IND is sent to the application with a CIP value of 0. In this case the application may evaluate the parameters Bearer Capability, Low Layer Compatibility and High Layer Compatibility to decide whether it is compatible to the call or not. Examples:

Service	Bits to be set in the CIP mask
Telephony Application	1 For calls within ISDN from equipment which does not send High Layer Compatibility info. 4 For calls from the analogue network. 16 For call within ISDN equipment which sends High Layer Compatibility info.
Fax Group 2/3 Application	4 For calls from the analogue network. 17 For calls within ISDN.
Non-standard 64 kBit/s data applications	2 No checking of High Layer Compatibility information is provided. The application should verify that no High Layer Compatibility information is received.
Non standard 56 kBit/s data applications	8 No checking of High Layer Compatibility information is provided. The application should verify that no High Layer Compatibility information is received.
Facsimile Group 4 application	2 For calls from equipment which does not send High Layer Compatibility information. The application should verify that no High Layer Compatibility information is received. 18 For call from equipment which sends High Layer Compatibility information.

This information element appears in:

LISTEN_REQ

6.8.17 Connected Number

Connected Number (struct)

The purpose of the parameter c*onnected number* information element is to indicate which number is connected to a call. The information element shall be coded according to ETS 300 097 [21].

Byte 0	Type of number and numbering plan identification (byte 3 of the connected number information element, see ETS 300 097 [21]). In the direction application to Profile B, the value supplied by the application shall be transmitted over the network, **0x00** is the suggested default value. In the direction Profile B to application, the value received from the network shall be passed to the application. The extension bit shall always be cleared.
Byte 1	Presentation and screening indicator (byte 3a of the connected number information element). In the direction application to Profile B, the value supplied by the application shall be transmitted over the network, **0x80** is the suggested default value. In the direction Profile B to application, the value received from the network shall be passed to the application. If this byte was not transmitted over the network, the controller provides the value **0x80** (user provided, not screened).
Bytes 2..n	Number digits of the connected number information element.

This information element appears in:

CONNECT_ACTIVE_IND

CONNECT_RESP

6.8.18 Connected Subaddress

Connected Subaddress (struct)

The purpose of the parameter *connected subaddress* information element is to identify the subaddress of the connected user of a call. The information element is coded according to ETS 300 097 [21].

Byte 0	Type of subaddress At the calling side the value supplied by application shall be transmitted over the network, **0x80** is the suggested default value (NSAP according X.213 [6]). In this case, the first subaddress information octet should have the value **0x50**. At the called side, the value received from the network shall be passed to the application.
Bytes 1..n	Contents of the connected subaddress information element.

This information element appears in:

CONNECT_ACTIVE_IND

CONNECT_RESP

6.8.19 Controller

Controller (dword)

The purpose of the parameter *controller* is to address a hardware unit, that gives access to an ISDN at the application's disposal. A *controller* supports none, one or several physical and logical connections. The parameter *controller* is a dword (to be compatible in size with PLCI and NCCI) with the range from 1 to 127 (0 reserved). Bit 7 additionally contains the information, if the message is used for internal (0) or external (1) equipment. Controllers are numbered sequentially and can be designed to handle external equipment additional to internal functionality or exclusively provide access to external equipment. External equipment e.g. is a handset.

Definition of external equipment behaviour, e.g. B-channel handling, is not covered by Profile B.

Format for controller:

0	0	0	Ext./Int.	Controller	
31	16	8	7	6	0

This information element appears in:

CONNECT_REQ

FACILITY_REQ

FACILITY_CONF

FACILITY_IND

FACILITY_RESP

LISTEN_REQ

LISTEN_CONF

MANUFACTURER_REQ

MANUFACTURER_CONF

MANUFACTURER_IND

MANUFACTURER_RESP

6.8.20 Data

Data (dword)

The purpose of the parameter *data* is to exchange a 32 bit pointer to the data area containing the information.

This information element appears in:

DATA_B3_REQ

DATA_B3_IND

6.8.21 Data Length

Data Length (word)

The purpose of the parameter *data length* is to specify the length of the data.

This information element appears in:

DATA_B3_REQ

DATA_B3_IND

6.8.22 Data Handle

Data Handle (word)

The purpose of the parameter *data handle* is to identify the data area in data exchange messages.

This information element appears in:

DATA_B3_REQ

DATA_B3_CONF

DATA_B3_IND

DATA_B3_RESP

6.8.23 Facility Selector

Facility Selector (word)

The purpose of the parameter *facility selector* is to identify the requested Profile B facility.

The defined values are:

0 Handset (external ISDN equipment) support
1 DTMF (Dual Tone Multi Frequency)

This information element appears in:

FACILITY_REQ

FACILITY_CONF

FACILITY_IND

FACILITY_RESP

6.8.24 Facility Request Parameter

Facility Request Parameter (struct)

The purpose of the parameter *facility request parameter* is to offer additional information concerning the message FACILITY_REQ.

This parameter is coded depending on *facility selector* as a structure with following elements:

Facility selector:

0 Parameter does not apply (coded as empty structure)
1 DTMF (Dual Tone Multi Frequency):

Function	word	1: Start DTMF listen on B-channel data 2. Stop DTMF listen 3: Send DTMF digits 4 to n: Reserved
Tone-Duration	word	Time in ms for one digit, default is 40 ms
Gap-Duration	word	Time in ms between the digits, default is 40 ms
DTMF-Digits	struct	Characters to be sent, coded as IA5-char. '0' to '9', '*', '#', 'A', 'B', 'C' or 'D', each character generates a unique DTMF- Tone.

Sending of DTMF characters shall interrupt the transmission of **DATA_B3_REQ**. After DTMF generation, the data transmission shall be resumed

This information element appears in:

FACILITY_REQ

6.8.25 Facility Confirmation Parameter

Facility Confirmation Parameter (struct)

The purpose of the parameter *facility confirmation parameter* is to offer additional information concerning the message FACILITY_CONF.

This parameter is coded depending on *facility selector* as a structure with following elements:

Facility selector:

0 **Parameter does not apply (coded as structure with a length of 0)**
1 **DTMF (Dual Tone Multi Frequency):**

DTMF information	word	0: sending of DTMF info successfully initiated 1: incorrect DTMF digit 2: unknown DTMF request

This information element appears in:

FACILITY_CONF

6.8.26 Facility Indication Parameter

Facility Indication Parameter (struct)

The purpose of the parameter *facility indication parameter* is to offer additional information concerning the message FACILITY_IND.

This parameter is coded depending on *facility selector* as a structure with following elements:

Facility selector:

0 **Handset Support:**

handset digits	byte array	Received characters, coded as IA5-char. '0' to '9', '*', '#', 'A', 'B', 'C' or 'D'; or '+': Handset off-hook '-': Handset on-hook

Facility selector:

1 **DTMF (Dual Tone Multi Frequency):**

DTMF digits	byte array	Received characters, coded as IA5-char. '0' to '9', '*', '#', 'A', 'B', 'C' or 'D'

This information element appears in:

FACILITY_IND

6.8.27 Facility Response Parameter

Facility Response Parameter (struct)

The purpose of the parameter *facility respond parameter* is to offer additional information concerning the message FACILITY_RESP.

This parameter is coded depending on *facility selector* as a structure with following elements:

Facility selector:

0 **Parameter does not apply (coded as structure with a length of 0)**
1 **Parameter does not apply (coded as structure with a length of 0)**

This information element appears in:

FACILITY_RESP

6.8.28 Flags

Flags (word)

The purpose of the parameter *flags* is to exchange additional protocol dependent information about the data.

Bit 0	qualifier bit
Bit 1	more data bit
Bit 2	delivery confirmation bit
Bit 3	expedited data bit
Bit 15	framing error bit, data may be invalid (only with corresponding B2 protocol)

This information element appears in:

DATA_B3_REQ

DATA_B3_IND

6.8.29 HLC

HLC (struct)

The purpose of the parameter *High Layer Compatibility (HLC)* information element is to provide a means which should be used by the remote user for compatibility checking. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

This information element appears in:

CONNECT_IND

CONNECT_REQ

6.8.29 Info

Info (word)

The purpose of the parameter *info* is to provide error information to the application. For each error which can be detected by the controller a unique code is defined, independing from the context of the error.

Profile B shall not generate other information values as defined below. In case of future extension of possible information values however an application should interpret any information value except class **0x00xx** as an indication that the corresponding request was rejected from Profile B. Class **0x00xx** indicates the successful handling of the corresponding request and returns additional information.

class 0x00xx: informative values (corresponding message was processed)

Value	Reason
0	request accepted
0x0001	NCPI not supported by current protocol, NCPI ignored
0x0002	flags not supported by current protocol, flags ignored
0x0003	alert already sent by another application

class 0x10xx: error information concerning CAPI_REGISTER

Value	Reason
0x1001	too many applications
0x1002	logical block size too small, shall be at least 128 bytes
0x1003	buffer exceeds 64 kByte
0x1004	message buffer size too small, shall be at least 1 024 bytes
0x1005	max. number of logical connections not supported
0x1006	reserved
0x1007	the message could not be accepted because of an internal busy condition
0x1008	OS Resource error (e.g. no memory)
0x1009	Profile B not installed
0x100A	Controller does not support external equipment
0x100B	Controller does only support external equipment

class 0x11xx: error information concerning message exchange functions

Value	Reason
0x1101	illegal application number
0x1102	illegal command or subcommand or message length less than 12 octets
0x1103	the message could not be accepted because of a queue full condition. The error code does not imply that Profile B cannot receive messages directed to another controller, PLCI or NCCI.
0x1104	queue is empty
0x1105	queue overflow, a message was lost. This indicates a configuration error. The only recovery from this error is to perform a CAPI_RELEASE.
0x1106	unknown notification parameter
0x1107	the message could not be accepted because of an internal busy condition
0x1108	OS Resource error (e.g. no memory)
0x1109	Profile B not installed
0x110A	Controller does not support external equipment
0x110B	Controller does only support external equipment

class 0x20xx: error information concerning resource/coding problems

Value	Reason
0x2001	message not supported in current state
0x2002	illegal Controller/PLCI/NCCI
0x2003	out of PLCI
0x2004	out of NCCI
0x2005	out of LISTEN
0x2006	out of FAX resources (protocol T.30 [14])
0x2007	illegal message parameter coding

class 0x30xx: error information concerning requested services

Value	Reason
0x3001	B1 protocol not supported
0x3002	B2 protocol not supported
0x3003	B3 protocol not supported
0x3004	B1 protocol parameter not supported
0x3005	B2 protocol parameter not supported
0x3006	B3 protocol parameter not supported
0x3007	B protocol combination not supported
0x3008	NCPI not supported
0x3009	CIP Value unknown
0x300A	flags not supported (reserved bits)
0x300B	facility not supported
0x300C	data length not supported by current protocol
0x300D	reset procedure not supported by current protocol

This information element appears in:

CONNECT_B3_CONF

CONNECT_CONF

INFO_CONF

DATA_B3_CONF

DISCONNECT_B3_CONF

DISCONNECT_CONF

LISTEN_CONF

RESET_B3_CONF

SELECT_B_PROTOCOL_CONF

6.8.30 Info Element

Info Element (struct)

The purpose of the parameter *info element* depends on the value of the parameter info number.

If the info number specifies an information element, the *info element* contains that information element with the coding as defined in ETS 300 102-1 [2]/Q.931 [8].

If the info number specifies a charging information *info element* contains a dword indicating the sum of charges accumulated by the network up to this moment.

If the info number specifies a message type the *info element* is an empty Profile B struct.

This information element appears in:

INFO_IND

6.8.31 Info Mask

Info Mask (dword)

The parameter *info mask* specifies which type of information for a physical connection or controller shall be provided by Profile B. The selected information shall be indicated within the message INFO_IND to the application. A given *info mask* (set in LISTEN_REQ) is valid until it is superseded by another LISTEN_REQ and applies to all information concerning the corresponding application. The *info mask* is coded as a bit field. A bit set to 1 means that corresponding INFO_IND messages shall be generated, a bit set to 0 means the specified information shall be suppressed. In the default *info mask* all bits are set to 0. If an application wants to change this value it shall send a LISTEN_REQ message even if it does not want to be informed about incoming calls.

Bit 0	Cause; cause information given by the net during disconnection. The parameter info element of the corresponding INFO_IND message is a Profile B struct which contains the cause information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.5.12).
Bit 1	Date/time; date/time information indicated by the net. The parameter info element of the corresponding INFO_IND message contains the date/time information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.6.1).
Bit 2	Display; display information to be displayed to the user. The parameter info element of the corresponding INFO_IND message contains the display information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.5.15).
Bit 3	User-user; user-user information that is transparently carried by the net. The parameter info element of the corresponding INFO_IND message contains the user-user information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.5.29).
Bit 4	Call progression; information referring to the progress of the call. There are five different INFO_IND messages that correspond to this information type, each with a unique info number. The first indication contains the information element progress indicator as defined in ETS 300 102-1 [2] and Q.931 [8]. The other four messages indicate the occurrence of the network events SETUP ACKNOWLEDGE, CALL PROCEEDING, ALERTING and PROGRESS. In these cases the parameter info number indicates the corresponding message type and the information element is an empty Profile B struct.
Bit 5	Facility; facility information to indicate the invocation and operation of supplementary services. The parameter info element of the corresponding INFO_IND message contains the facility information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.6.2).
Bit 6	Charging information; connection oriented charging information provided by the net. There are two different INFO_IND messages with unique info numbe*r* values that correspond to this information type. The first one shows the sum of charging units indicated by the net up to this moment, the second the sum of charges in the national currency indicated by the net up to this moment. In both cases the parameter info element is coded as a Profile B struct containing a dword. It is highly recommended to provide only one of this two types of charging information to the user and to transform one type to the other. However, in some networks this might be impossible due to the information provided from the net. In these cases it is not defined, if the current charges are represented by only one or both or the sum of this indicated charges.
Bit 7	Called Party number; identifies the called party of a call. The parameter info element of the corresponding INFO_IND message contains the called party number information element defined in ETS 300 102-1 [2] and Q.931 [8] (both 4.5.8).
Bits 7-31	Reserved, shall be set to 0

This information element appears in:

LISTEN_REQ

6.8.32 Info Number

Info Number (word)

The purpose of the parameter *info number* specifies the coding of the parameter *info element* and the type of information which is carried by this INFO_IND message. The high byte is structured as a bit field and indicates which type of information is held in the low byte.

Bit 15	If this bit set to 1 the low byte contains a message type, if it is set to 0 the low byte represents an information element type.
Bits 14	If this bit is set to 1 the low byte indicates supplementary information not covered by network events or information elements. In this case bit 15 shall be set to 0.
Bits 13-8	Reserved, set to 0.

If bit 15 is set, the low byte containing the message type is coded according to ETS 300 102-1 [2]/Q.931 [8]. In this case the INFO_IND message indicates the occurrence of a network event according to the specified message and the parameter *info element* is an empty Profile B struct.

If bits 14 and 15 are cleared, the low byte represents an information element type coding according to ETS 300 102-1 [2]/Q.931 [8]. The parameter *info element* contains the content of the information element.

If bit 14 is set, the low byte represents supplementary information. The defined values are

0 sum of charges in charging units. In this case the parameter *info element* contains the content of the information element.

1 sum of charges in national currency. In this case the parameter *info element* contains the content of the information element.

This information element appears in:

INFO_IND

6.8.33 LLC

LLC (struct)

The purpose of the parameter *Low Layer Compatibility (LLC)* information element is to provide a means which should be used for compatibility checking by an addressed entity (e.g. a remote user or an inter working unit or a high layer function network node addressed by the calling user). The *Low Layer Compatibility* information element is transferred transparently by ISDN between the call originating entity (e.g. the calling user) and the addressed entity. If *Low Layer Compatibility* negotiation is allowed by the network, the *Low Layer Compatibility* information element is also passed transparently from the addressed entity to the originating entity. The information element is coded according to ETS 300 102-1 [2]/Q.931 [8].

This information element appears in:

CONNECT_ACTIVE_IND

CONNECT_IND

CONNECT_REQ

CONNECT_RESP

6.8.34 Manu ID

Manu ID (dword)

The purpose of the parameter *Manu ID* is to exchange a dword inside MANUFACTURER**-**Messages which identifies the manufacturer. Every manufacturer offering MANUFACTURER-Messages should choose a unique value (e.g. shortcut of company name).

This information element appears in:

MANUFACTURER_REQ

MANUFACTURER_RESP

MANUFACTURER_CONF

MANUFACTURER_IND

6.8.35 Manufacturer Specific

Manufacturer Specific

The purpose of the parameter *manufacturer Specific* is to exchange manufacturer specific information.

This information element appears in:

MANUFACTURER_REQ

MANUFACTURER_RESP

MANUFACTURER_CONF

MANUFACTURER_IND

6.8.36 NCCI

NCCI (dword)

The purpose of the parameter *NCCI* is to identify a logical connection. The *NCCI* is given by Profile B during creation of the logical connection. Depending on the layer 3 protocol selection (e.g. ISO 8208 [3]), it is possible to have multiple *NCCIs* based on one PLCI. The *NCCI* is a dword with a range from 1 to 65 535 (0 reserved), coded as described below, and includes additionally the corresponding PLCI and controller.

Format for NCCI:

NCCI		PLCI	Ext./Int.	Controller	
31	16	8	7	6	0

This information element appears in:

CONNECT_B3_ACTIVE_IND

CONNECT_B3_ACTIVE_RESP

CONNECT_B3_CONF

CONNECT_B3_IND

CONNECT_B3_RESP

DATA_B3_CONF

DATA_B3_IND

DATA_B3_REQ

DATA_B3_RESP

DISCONNECT_B3_CONF

DISCONNECT_B3_IND

DISCONNECT_B3_REQ

DISCONNECT_B3_RESP

FACILITY_REQ

FACILITY_CONF

FACILITY_IND

FACILITY_RESP

RESET_B3_CONF

RESET_B3_IND

RESET_B3_REQ

RESET_B3_RESP

6.8.37 NCPI

NCPI (struct)

The purpose of the parameter *NCPI* is to provide additional protocol specific information.

For the layer 3 protocols ISO 8208 [3] and X.25 the parameter data of structure *NCPI* are coded as follows:

Byte 0	Bit field
	[0]: Enable the usage of the delivery confirmation procedure in call set-up and data packets (D-Bit).
	[1..7]: Reserved.
Byte 1	Logical channel group number of the permanent virtual circuit (PVC) to be used. In the case of virtual calls (VC) this number shall be set to zero.
Byte 2	Logical channel number of the permanent virtual circuit (PVC) to be used. In the case of virtual calls (VC) this number shall be set to zero.
Bytes 3..n	Bytes following the packet type identifier field in the X.25 PLP packets.

For layer 3 protocol T.30 [14] (fax group 3) the parameter data of structure *NCPI* are valid only for DISCONNECT_B3_IND and coded as follows (in every other message the structure is empty):

word	Rate	actual used bit rate, coded as unsigned integer value
word	resolution	0: standard 1: high
word	format	0: SFF (Default, description in annex A) 1: Plain FAX Format (modified Huffman coding) 2: PCX 3: DCX 4: TIFF 5: ASCII 6: Extended ANSI 7: Binary-File transfer
word	pages	number of pages, coded as unsigned integer value
struct	receive id	id of remote side

This information element appears in:

CONNECT_B3_ACTIVE_IND

CONNECT_B3_T90_ACTIVE_IND

CONNECT_B3_IND

CONNECT_B3_REQ

CONNECT_B3_RESP

DISCONNECT_B3_IND

DISCONNECT_B3_REQ

RESET_B3_REQ

RESET_B3_RESP

6.8.38 PLCI

PLCI (dword)

The purpose of the parameter *PLCI* is to describe a physical connection between two endpoints. The *PLCI* is given by Profile B during creation of the physical connection. The *PLCI* is a dword with the range from 1 to 255 (0 reserved), coded as described below, and additionally includes the controller.

Format for PLCI:

0	0	PLCI	Ext./Int.	Controller	
31	16	8	7	6	0

This information element appears in:

CONNECT_ACTIVE_IND

CONNECT_ACTIVE_RESP

CONNECT_B3_REQ

CONNECT_CONF

CONNECT_IND

CONNECT_RESP

DISCONNECT_REQ

DISCONNECT_CONF

DISCONNECT_IND

DISCONNECT_RESP

FACILITY_REQ

FACILITY_CONF

FACILITY_IND

FACILITY_RESP

INFO_REQ

INFO_CONF

INFO_IND

INFO_RESP

SELECT_B_PROTOCOL_REQ

SELECT_B_PROTOCOL_CONF

6.8.39 Reason

Reason (word)

The purpose of the parameter *reason* is to provide error information to the application regarding the clearing of a physical connection. The defined values are:

0	normal clearing, no cause available
0x3301	protocol error layer 1
0x3302	protocol error layer 2
0x3303	protocol error layer 3
0x3304	another application got that call (see LISTEN_REQ)
0x34xx	disconnect cause from the network according to ETS 300 102-1 [2]/Q.931 [8]. In the field 'xx' the cause value received within a cause information element (octet 4) from the network is indicated.

This information element appears in:

DISCONNECT_IND

6.8.40 Reason_B3

Reason_B3 (word)

The purpose of the parameter *reason* is to provide error information to the application regarding the clearing of a logical connection. The defined values are:

protocol independent:

0	normal clearing, no cause available
0x3301	protocol error layer 1 (broken line or B-channel removed by signalling protocol)
0x3302	protocol error layer 2
0x3303	protocol error layer 3

T.30 [14] specific reasons:

0x3311	connecting not successful (remote station is no fax G3 machine)
0x3312	connecting not successful (training error)
0x3313	disconnected before transfer (remote station does not support transfer mode, e.g. resolution)
0x3314	disconnected during transfer (remote abort)
0x3315	disconnected during transfer (remote procedure error (e.g. unsuccessful repetition of T.30 [14] commands)
0x3316	disconnected during transfer (local tx data underrun)
0x3317	disconnected during transfer (local rx data overflow)
0x3318	disconnected during transfer (local abort)
0x3319	illegal parameter coding (e.g. SFF coding error)

6.8.41 Reject

Reject (word)

The purpose of the parameter *reject* is to define the action of Profile B for incoming calls.

The defined values are:

0	Accept the call
1	Ignore the call
2	reject call, normal call clearing
3	reject call, user busy
4	reject call, requested circuit/channel not available
5	reject call, facility rejected
6	reject call, channel unacceptable
7	reject call, incompatible destination
8	reject call, destination out of order

This information element appears in:

CONNECT_B3_RESP

CONNECT_RESP

6.9 State diagram

6.9.1 User's guide

To explain the message exchange between Profile B and application, a graphic description is mandated. In the absence of an international standard for the description of a message exchange between two local entities, a new way of presentation was created. The state machines on the following pages are described in the form of a state diagram covering application and controller. This state diagram is a monitor view of an idealised interface. In reality the Profile B is not only an interface definition, it is also a concrete instantiation.

The state diagram on the following pages is split into three separate state machines:

1. LISTEN state machine

2. PLCI state machine (physical connections)

3. NCCI state machine (logical connections)

On every physical connection, identified by a PLCI, several logical layer 3 links could exist, identified by a NCCI. Therefore a splitting into PLCI and NCCI state machine is necessary. A description of "n" physical links with "m" logical links at one time in one state machine is impossible. Therefore only one PLCI or one NCCI at a time is considered in the state machine.

Profile B messages LISTEN_REQ and LISTEN_CONF are described in a separate state machine, because the availability of a successful LISTEN setting exceeds the lifetime of logical and/or physical connections.

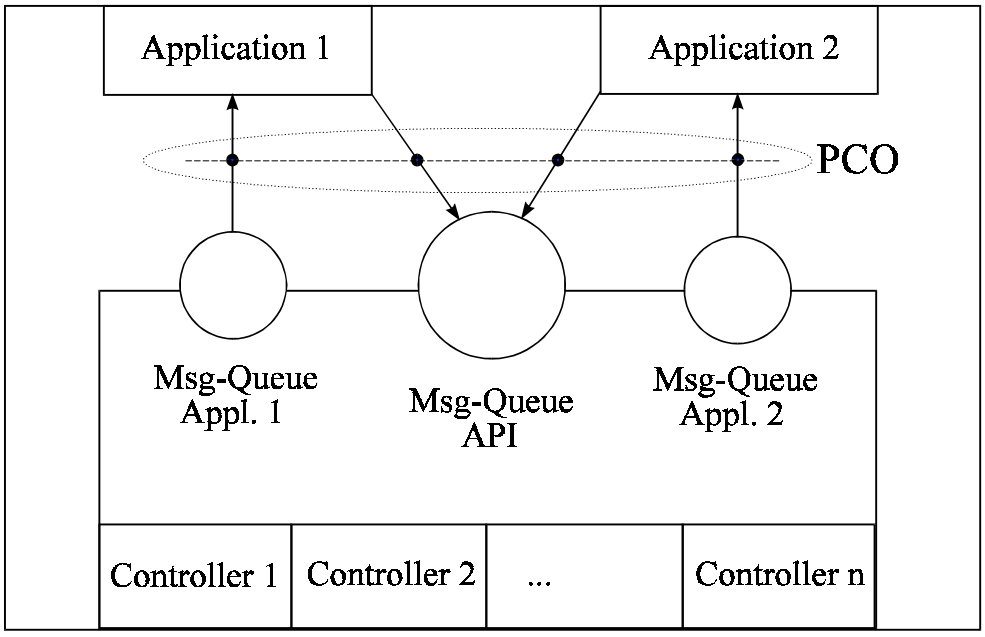


Figure 33: Position of Point of Control and Observation (PCO)

6.9.2 Explanation

The state diagrams define a faultless exchange of messages. The PCO for the message exchange description is on the level of the Profile B operations. For real implementations it is not allowed that an asynchronous exchange of messages results in an error condition.

The state diagrams define the flow of the messages on the PCO without consideration of their possible asynchronicity in real implementations.

Confirmations and responses, which do not evoke a state transition, are not shown in this state diagrams.

In "ANY-State" it is allowed that an expected confirmation on a request or an expected response appears.

The messages MANUFACTURER_REQ, MANUFACTURER_CONF, MANUFACTURER_IND and MANUFACTURER_RESP could result in incompatibility. They are not described in the state diagrams.

Requests with an invalid PLCI or an invalid NCCI are wrong messages and therefore are not described in the state diagrams.

INFO_REQ and INFO_IND are network specific elements which can appear at any time. The use of INFO_REQ especially for "overlap sending" is described in the PLCI-state machine 1/2.

FACILITY_REQ, FACILITY_CONF, FACILITY_IND and FACILITY_RESP are facility specific messages which can appear at any time. Therefore they can occur in every state of the LISTEN-, PLCI- and NCCI-state machine. Especially the FACILITY_IND concerning "Handset Support" is described in the PLCI-state machine 1/2. The flow of the messages for the Handset Support depends on the real handset interface (e.g. Additional Equipment Interface (AEI)) or manufacturer specific codecs. So it is possible, that only a part of the described flow of the messages for the Handset Support is used. But it is not allowed to use the FACILITY messages for the Handset Support in another way, as described in the message definition and the state machines.

LISTEN - state machine

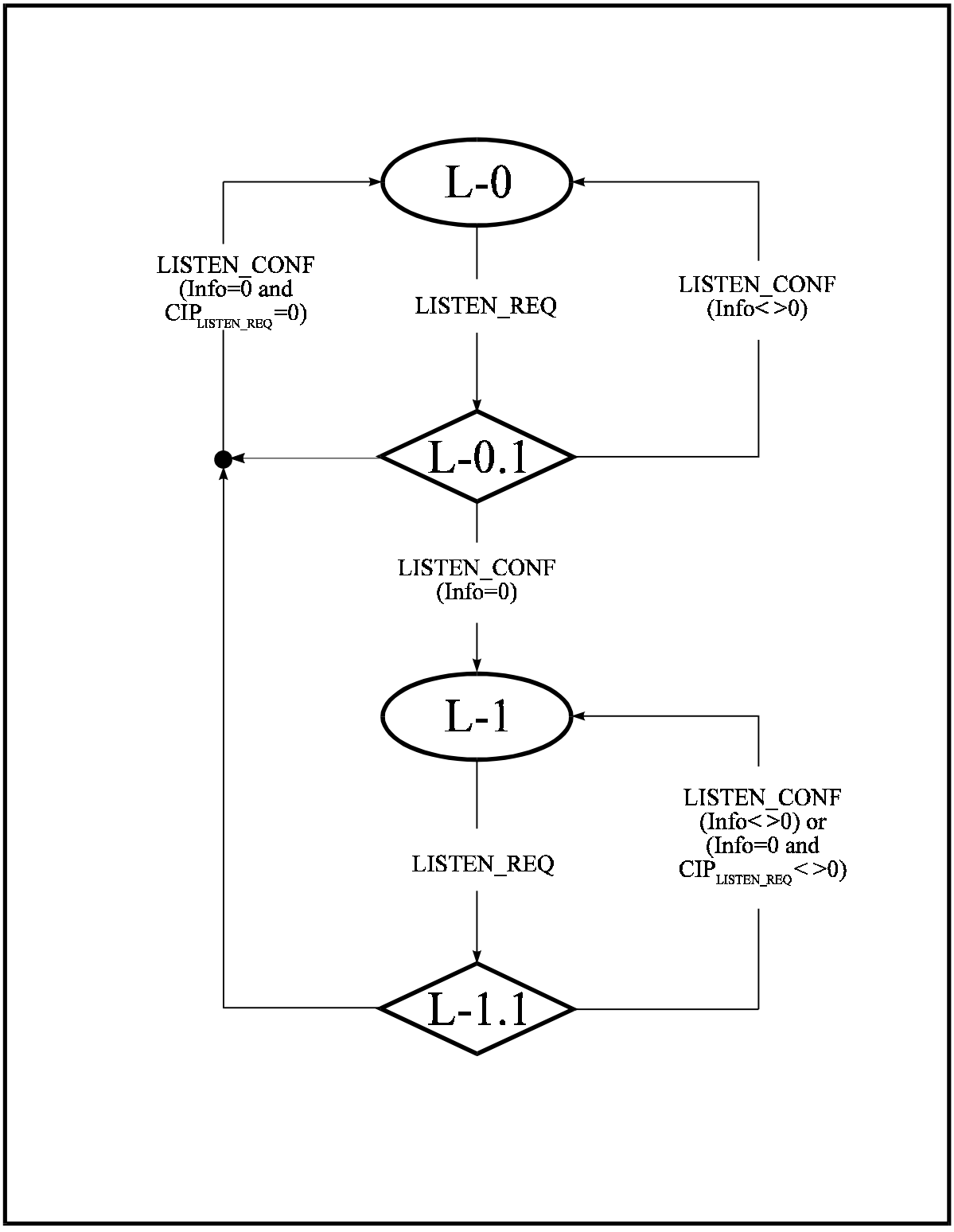


Figure 34: LISTEN -state machine

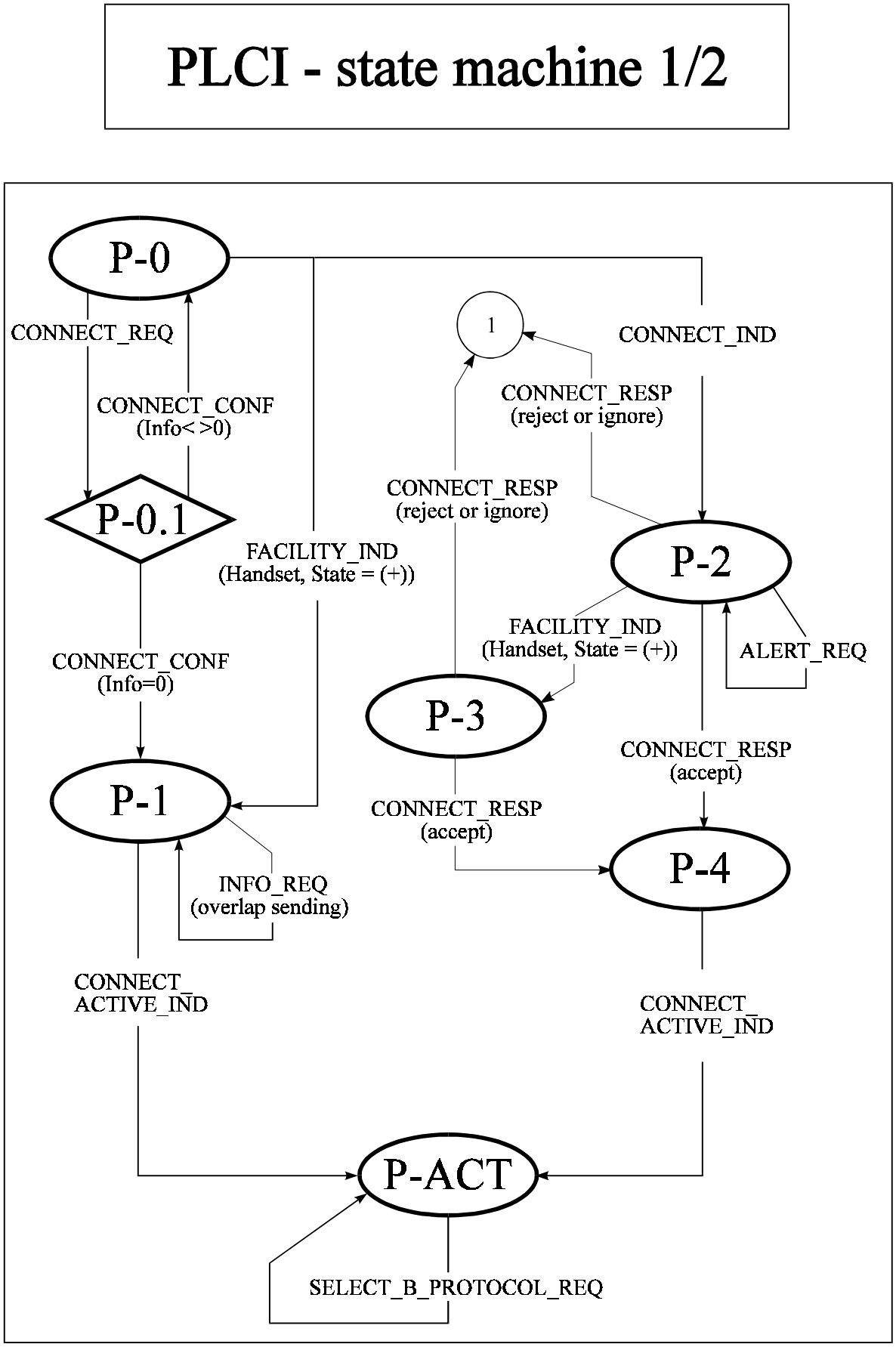


Figure 35: PLCI -state machine (1/2)

PLCI - state machine 2/2

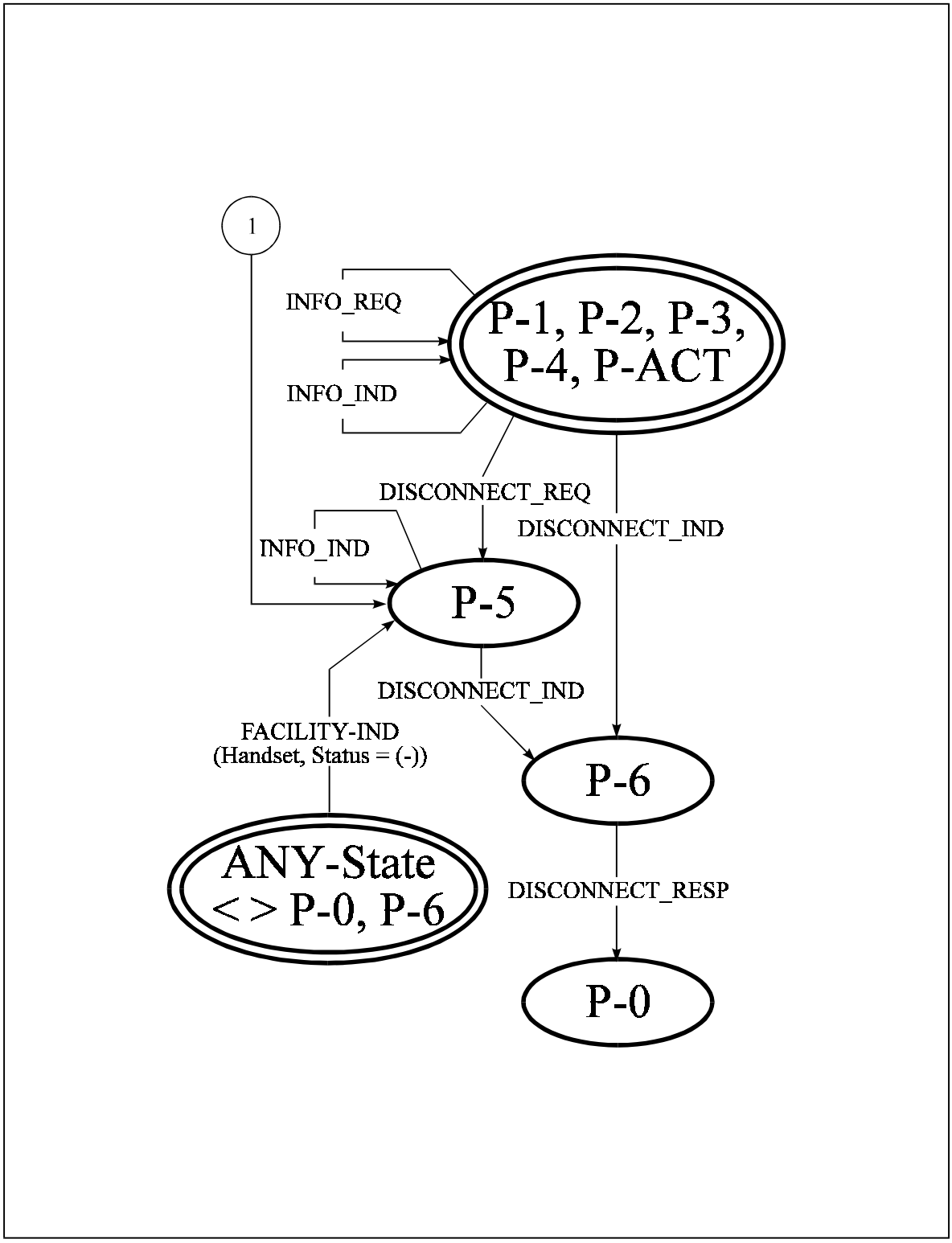


Figure 35: PLCI -state machine (2/2)

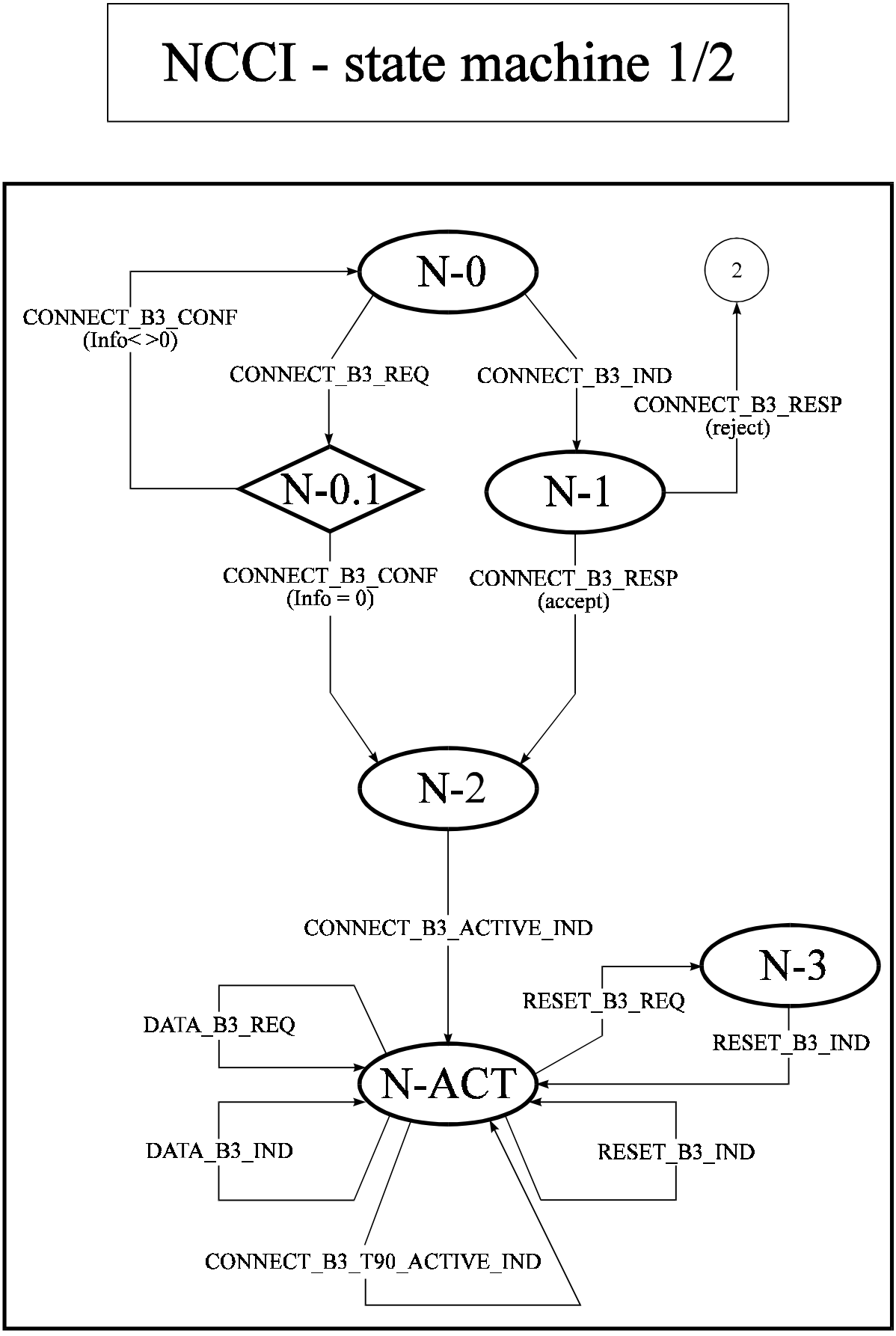


Figure 36: NCCI -state machine (1/2)

7 Operating system description

This clause describes the operating system specific implementation for both profiles.

7.1 DOS

7.1.1 DOS Operation System specific implementation for Profile A

This subclause describes the operating system specific implementation for the DOS operating system. For the following description, the base MS-DOS version is the version number 3.1.

A NAF implementation under DOS shall offer the functionality of the exchange functions described in a generic way in subclause 5.3.

In this subclause, the mapping and implementation of these functions are described on a function per function basis. For each function, a coding example in C language is given.

7.1.1.1 Introduction

Except for the function **PciGetHandles**, the implementation of the exchange method for DOS is based on a direct access mechanism. The access point is a far function address provided by the NAF. This function address is mapped to the generic type PCI_HANDLE.

To make sure that the function address provided by the NAF is correct, the PUF may cfheck a signature located in front of the function address before calling the NAF.

To perform this check, the PUF shall examine the memory area located just in front of the function address. There the signature is located, which shall contain the eight character constant "ISDN PCI". If this signature is available, the PUF assumes the NAF function address is correct.

Only one access point shall be provided by the NAF. A supplied parameter shall indicate the function to be invoked. This parameter is named **function code**.

Parameters are passed from the PUF to the NAF using the stack. The PUF shall ensure a minimum stack space of 128 bytes on call. When the NAF receives the control of the CPU, the first parameter on the stack is the function code, followed by parameters based on the particular function.

The function code is passed as a 2 byte integer value.

The NAF has to place the return code in the AX register. The NAF procedure is not in charge of cleaning the stack on return. The C call convention is used: the calling PUF pushes parameters right to left and restores the stack on return.

The alignment of the PCIMPB generic structure is **byte**.

7.1.1.2 Mapping of generic types and constants

Under DOS, the following mapping shall be used for the generic types described in the subclause 5.3:

Generic Type	DOS Mapping
PCI_INTEGER	2 byte integer (a word)
PCI_BYTEARRAY	far pointer (segment:offset address)
PCI_EXID	Unique identifier provided by NAF (2 byte integer)
PCI_HANDLE	far function address (segment: offset address)
PCI_PROCEDURE	far function address (segment: offset address)

As usual for DOS, all values are in little endian (low byte - high byte) order.

The **function code**, used to invoke the exchange functions, shall be assigned as follows:

Function	Function code value
PciGetProperty	1
PciRegister	2
PciDeregister	3
PciPutMessage	4
PciGetMessage	5
PciSetSignal	6

C presentation of these definitions looks as follows:

```
/*
 * Generic type mappings
 */
typedef short int           PCI_INTEGER;
typedef char far *          PCI_BYTEARRAY;
typedef short int           PCI_EXID;
typedef short int           (far *  PCI_HANDLE) ();
typedef void                (far *  PCI_PROCEDURE) ();

/*
 * Function code constants
 */
#define     PCIGETPROPERTY          1
#define     PCIREGISTER             2
#define     PCIDEREGISTER           3
#define     PCIPUTMESSAGE           4
#define     PCIGETMESSAGE           5
#define     PCISETSIGNAL            6

/*
 * Signature
 */
#define     PCISIGNATURE            'ISDN PCI'    /* multi characters constant */
```

7.1.1.3 Description of functions

The PUF is in charge to provide a minimal stack during a function call. The minimal stack size is 128 bytes.

In the description the access to one is described for simplicity in the coding examples. However the access of a PUF to multiple NAFs is not excluded.

7.1.1.3.1 PciGetHandles

Under DOS, the implementation of the **PciGetHandles** function shall use a character device driver named "PCIDD$" to retrieve the available PCI-Handles. This function call is the exception on the basic principle - direct access - under DOS.

The maximum theoretical amount of PCI-Handles which can be retrieved is 4 096. However, the implemented device driver will probably have a practical limit which lies far below and depends on the implementation of the device driver itself.

The following operation shall be performed by the PUF, in order:

- Open the "PCIDD$" character device driver.
- Prepare a buffer in memory, big enough to hold the maximum amount of PCI-Handles to be retrieved.
- Issue a IOCTL system read call: Receive control Data from Character Device.
  - BX shall contain the dos handle of the device driver,
  - CX shall contain the length of the memory buffer prepared above,
  - DS:DX shall point to the memory buffer.
- Check the success of the operation (check carry flag).
- In case of error, optionally issue a DOS Get Extended Error function call to receive a more comprehensive error code.
- On successful return, AX contains the number of bytes provided by the device driver, the buffer contains the available PCI-Handles in a row. The number of available PCI-Handles is calculated by dividing the AX value by 4, the size of a far address function pointer.
- Close the device driver.

C coding example:

```
...
#include    <dos.h>                 /* declarations for IOCTL call */
#include    <fcntl.h>               /* declarations for open mode */
...
#define     SUCCESS         0           /* No error */
#define     MAXHANDLES          64          /* max amount of handles to be read */
...
PCI_HANDLE PCIHandlesArray[MAXHANDLES]          /* buffer for receiving PCI-Handles */
...
PCI_INTEGER MaxHandles;                         /* max amount of handles to be read */
PCI_HANDLE far * PCIHandles;                    /* far pointer to buffer of PCI-Handles */
PCI_INTEGER far * ActualHandles;                /* far ptr to amount of PCI-Handles received */
{
int fildes;                                 /* file descriptor */
int error;
union _REGS regs;
struct _SREGS segregs;
struct _DOSERROR errorinfo;

/* open the driver */
if (_dos_open ("PCIDD$", _O_RDWR, &fildes) != SUCCESS)
        {
        /* device driver not accessible; perform error processing */
        error = ...
        }
else
        {
        /* prepare IOCTL read from device driver */
        _segread (&segregs);
        segregs.ds = FP_SEG (PCIHandles);       /* set-up segment address */
        regs.x.dx = FP_OFF (PCIHandles);        /* and offset */
        regs.x.cx = MaxHandles * sizeof(PCI_HANDLE);
        regs.x.bx = fildes;                     /* set dos file handle */
        regs.x.ax = 0x4402;                         /* IOCTL read from character device */

        /* issue IOCTL read from device driver */
        _intdosx (&regs, &regs, &segregs);

        /* close the driver */
        _dos_close (fildes);

        /* check for error */
        if (regs.x.cflag & 1)                   /* check processors carry flag */
                {
```

```
                /* error has occured; perform error processing */
                _dosexterr (&errorinfo);
                error = doserror.exterror;
                ...
                }
        else
                {
                /* Successfull operation. Set count of handles received */
                *ActualHandles = regs.x.ax / sizeof(PCI_HANDLE);
                error = SUCCESS;
                }
...
```

7.1.1.3.2 PciGetProperty

This function is in charge of retrieving the NAF-Property from the NAF. To issue the function call, the PUF shall possess the PCI-Handle of the NAF it wants to access. Before accessing the NAF, the PUF may check, if the PCI-Handle it uses is valid by checking the signature of the access point the PCI-Handle is pointing to.

The following operation shall be carried out by the PUF, in order:

- may examine memory area pointed to by the PCIHandle to find out if NAF is loaded and. check the signature for the character constant 'ISDN PCI' in that case;
- call the address with the PciGetProperty **function code** and the parameters provided by the PUF;
- check return code.

C coding example:

```
...
#include      <memory.h>              /* memory compare func declarations */
...
#define       SUCCESS         0       /* No error */
#define       PCIGETPROPERTY      1
#define       PCISIGNATURE    'ISDN PCI'
#define       SIGNATURESIZE   8
...
PCI_HANDLE PCIHandle;
PCI_INTEGER MaximumSize;
PCI_BYTEARRAY Property;
PCI_INTEGER far * ActualSize;
{
PCI_INTEGER error;
char far * signature;

signature = (char far *) PCIHandle - SIGNATURESIZE;
if (_fmemcmp (signature,PCISIGNATURE,SIGNATURESIZE) == SUCCESS)
        {
        /* signature is correct. call the entry point */
        error = (*PCIHandle) (PCIGETPROPERTY, MaximumSize, Property, ActualSize);
        ...
        }
else
        {
        /* signature wrong. process error */
        error = ...
...
```

7.1.1.3.3 PciRegister

This function is in charge of providing an association between a PUF and a NAF. To issue the function call, the PUF shall possess the PCI-Handle of the NAF it wants to access Before accessing the NAF, the PUF may check, if the PCI-Handle it uses is valid by checking the signature of the access point the PCI-Handle is pointing to.

For this function call, 2 structures shall be prepared by the PUF and shall be passed on the function stack. The first structure is the PCIRegisterInfo structure as declared in clause 5.3. The second is the operating system dependent PCIOpSysInfo structure, which for DOS has the following layout:

Element Name	Type	Validity	Explanation
MaxNCOCount	2 byte integer	on call	Shall be set to the maximum amount of NCOs the PUF intends to create during the association.
MaxPacketSize	2 byte integer	on call	Shall be set to the maximum size of a data packet the NAF shall accept on a user connection.
MaxPacketCount	2 byte integer	on call	Shall be set to the maximum amount of packets of the above size the NAF shall buffer per user connection.
AddBufferSize	4 byte integer	on call	If the PUF wants to provide buffer space to the NAF, it shall set this value to the size of the buffer space it donates. Otherwise the value shall be set to zero (0).
AddBufferSpace	far address (segment: offset)	on call	If the structure element AddBufferSize is non-zero, this element shall point (far) to the donated, additional buffer space.
BufferNeeded	4 byte integer	on return	In case the NAF has not enough buffer space available to guarantee the requested connection characteristics, the amount of additional buffer needed is returned into this element by the NAF.

The information provided with this structure helps the NAF to optimize its internal resources. Therefore, the information given by the PUF shall be carefully weighted. This is especially true in an environment, where a NAF serves several PUFs at the same time.

In the case a NAF has not available enough memory resources to fulfil the requested characteristics, the PciRegister function will fail and return a BuffersTooSmall error code. In this case the amount of buffer missing can be taken from the BufferNeeded element of the above structure.

On successful return of the PciRegister function, the Exchange-ID becomes available, which shall be used as a parameter on subsequent exchange mechanism function calls.

The following operation shall be carried out by the PUF, in order:

- examine memory area pointed to by the PCIHandle to find out if NAF is loaded. Check the signature for characters 'ISDN PCI';

- allocate and set-up the two structures PCIRegisterInfo and PCIOpSysInfo. The PCIOpSysInfo structure may optionally contain a pointer to additional buffer space which shall be donated to the NAF;

- call the exchange function with the PciRegister **function code** and the parameters provided by the PUF;

- check return code. If the return code indicates OutOfBuffers then the call may be repeated with correct adjusted buffer space to be donated to the NAF;

- KEEP the returned Exchange-ID for later calling.

C coding example:

```
...
#include    <memory.h>                  /* memory compare func declarations */
#include    <malloc.h>                  /* memory allocation functions */
...
```

```
#define     SUCCESS                 0           /* No error */
#define     PCIREGISTER                 2
#define     PCISIGNATURE                'ISDN PCI'
#define     SIGNATURESIZE               8
#define     E_OUT_OF_BUFFERS        148         /* BuffersTooSmall error code */
...
struct pci_register {                                   /* structure containing registering info */
            PCI_INTEGER     PUFVersion;     /* optional: give PUF version */
            PCI_INTEGER     PUFType;            /* optional: give PUF type */
            PCI_INTEGER     MaxMsgSize;         /* return: max size of a message */
};

struct pci_opsys {                                      /* structure containing registering info */
            short int       MaxNCOCount;        /* optional: give max count of NCOs */
            short int       MaxPacketSize;      /* optional: give expected max size and */
            short int       MaxPacketCount;     /* max count of packets to buffer */
            long int        AddBufferSize;      /* optional: give to NAF size and */
            void far *      AddBufferSpace;     /* pointer to additional buffer */
            long int        BufferNeeded;       /* return: amount of add buffer needed */
};
...

/*
  * before calling the PCIRegister function further down, allocate and prepare the structures
 * requested by this function call
 */
struct pci_register PCIRegisterInfo {
            2,              /* Set PUF version to 2, equaling current ETS Version*/
            0,              /* Set PUF type to 0 as indicated in [2]*/
            0               /* Initialise (expected) return value of MaxMsgSize */
};

struct pci_opsys PCIOpSysInfo {
            2,              /* Set max amount NCOs PUF intends to create */
            1024,           /* Set max size of data packets NAF shall accept */
            8,              /* Set max count of packets NAF shall buffer per NCO */
            0,              /* Set size of memory PUF wants to donate to NAF */
            (void far *) NULL,      /* Set pointer to (donated) buffer space */
            0               /* Initialise (expected) return value of BufferNeeded */
};
...
PCI_HANDLE PCIHandle;
struct pci_register far * RegisterStruct;;
PCI_EXID far * ExchangeID
{
PCI_INTEGER error;
char far * signature;
void far * buffer;

signature = (char far *) PCIHandle - SIGNATURESIZE;
if (_fmemcmp (signature,PCISIGNATURE,SIGNATURESIZE) != SUCCESS)
        {
        /* signature wrong. process error */
        error = ...
        }
else
        {
        /* signature is correct. call the entry point */
        error = (*PCIHandle) (PCIREGISTER, &PCIRegisterInfo, &PCIOpSysInfo, ExchangeID);
        if (error == E_OUT_OF_BUFFERS)
                {
                /* NAF needs more buffer space; try to allocate */
                buffer = _fmalloc ((size_t) PCIOpSysInfo.BufferNeeded);
```

```
                if (buffer)
                        {
                        /* there is buffer, so it´s worth another try; adjust PCIOpSysInfo structure */
                        PCIOpSysInfo.AddBufferSize = PCIOpSysInfo.BufferNeeded;
                        PCIOpSysInfo.AddBufferSpace = buffer;
                        PCIOpSysInfo.BufferNeeded = 0;
                        /* call PciRegister again ... */
                        }
                }
        error=(*PCIHandle)(PCIREGISTER,&PCIRegisterInfo,&PCIOpSysInfo,ExchangeID);
        if (error)
                {
                /* Process error */
                ...
...
```

7.1.1.3.4 PciDeregister

This function is in charge to disassociate a PUF and a NAF.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciDeregister **function code** and the Exchange-ID related to the current association;

- check return code.

C coding example:

```
...
#define       PCIDEREGISTER           3
...
PCI_HANDLE PCIHandle;
PCI_EXID ExchangeID;
{
PCI_INTEGER error;

/* call the entry point */
error = (*PCIHandle) (PCIDEREGISTER, ExchangeID);
...
```

7.1.1.3.5 PciPutMessage

This function is in charge to provide a message from a PUF to a NAF. Parameters shall be provided in the same order as indicated in the generic description of the PciPutMessage function.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciPutMessage **function code** and the Exchange-ID related to the current association as well as the correct set-up PCI Message Parameter Block and the associated buffers;

- check return code.

C coding example:

```
...
#define       PCIPUTMESSAGE                   4
...
struct pci_mpb {
                PCI_INTEGER     MessageID;
                PCI_INTEGER     MessageMaximumSize;
```

```
                PCI_INTEGER     MessageActualUsedSize;
                PCI_INTEGER     DataMaximumSize;
                PCI_INTEGER     DataActualUsedSize;
};
...
PCI_HANDLE PCIHandle;
PCI_EXID ExchangeID;
struct pci_mpb far * PCIMbp;
PCI_BYTEARRAY Message;
PCI_BYTEARRAY Data;
{
PCI_INTEGER error;

/* call the entry point */
error = (*PCIHandle) (PCIPUTMESSAGE, ExchangeID, PCIMbp, Message, Data);
...
}
```

7.1.1.3.6 PciGetMessage

This function is in charge to provide the PUF with a message coming from the NAF. Parameters shall be provided in the same order as indicated in the generic description of the PciGetMessage function.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciGetMessage **function code** and the Exchange-ID related to the current association as well as the correct set-up PCI Message Parameter Block and the associated buffers;

- check return code.

C coding example:

```
...
#define     PCIGETMESSAGE               5
...
struct pci_mpb {
                PCI_INTEGER     MessageID;
                PCI_INTEGER     MessageMaximumSize;
                PCI_INTEGER     MessageActualUsedSize;
                PCI_INTEGER     DataMaximumSize;
                PCI_INTEGER     DataActualUsedSize;
};
...
PCI_HANDLE PCIHandle;
PCI_EXID ExchangeID;
struct pci_mpb far * PCIMbp;
PCI_BYTEARRAY Message;
PCI_BYTEARRAY Data;
{
PCI_INTEGER error;

/* call the entry point */
error = (*PCIHandle) (PCIGETMESSAGE, ExchangeID, PCIMbp, Message, Data);
...
}
```

7.1.1.3.7 PciSetSignal

This function is in charge to provide the NAF with the address of a function located inside the PUF, which shall be called-back if a message becomes available for the PUF.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciSetSignal **function code** and the Exchange-ID related to the current association as well as the correct set-up function address of the call-back routine;

- check return code.

C coding example:

```
#define        PCISETSIGNAL            6
/* CallBack function called in interrrupt context */
void far CallBackFunc ()
{
...
return;
}


/*
 * Code to set up the notification process
 */
...
PCI_HANDLE PCIHandle;
PCI_EXID ExchangeID;
{
PCI_INTEGER error;

/* call the entry point */
error = (*PCIHandle) (PCISETSIGNAL, ExchangeID, &CallBackFunc);
...
}
```

The NAF calls back the PUF with the following conventions applying:

- the NAF provides a minimal stack size of 128 bytes;
- the values of the DS and ES segments are undefined;
- interrupts are disabled.

Gained control, the PUF:

- may or may not enable interrupts;

- is allowed call the NAF via the PciGetMessage or the PciPutMessage function;

- shall not invoke other exchange function calls besides the PciGetMessage and the PciPutMessage functions;

- shall not issue DOS system calls;

- shall not let interrupts be disabled over an extended period of time and shall return from the call-back function as quick as possible.

The NAF called via the PciGetMessage or the PciPutMessage function may enable interrupts. However, the NAF shall not call the call-back routine again, until the call back routine has returned normally.

At the end of the call back routine the PUF shall return to the NAF. Only the SS:SP register pair shall be preserved by the PUF.

7.1.1.4 Availability of NAF's PCI_HANDLE

To be accessible via the PciGetHandles function call, a NAF shall issue a declaration action. The inverse action, extraction from the list of available NAFs, is described too. These actions are operating system specific.

7.1.1.4.1 Declaration action

Under DOS, the NAF uses the PCIDD$ Device Driver to declare itself, issuing an IOCTL write command, passing a structure containing the action code (Declare) and the handle of the NAF.

The maximum number of NAF than the 'PCIDD$' Device Driver can register is 32.

The following operation will take place in order:

- open the 'PCIDD$' driver;
- prepare the following structure:
  - one word: command code, 0 x 4544 (characters 'DE', DEclaration);
  - one double-word: address of the NAF entry point.

- issue a IOCTL system call write command:
  - CX contains the size of the declaration structure (6);
  - DS:DX point to the structure.

- check the success of the operation (check CARRY FLAG);
- in case of error, issue a Get Extended Error function call to get a more comprehensive error code;
- close the driver.

The command will end successfully even if the NAF is already declared. In this case, no action takes place.

The command gives an error on the following cases. In these cases, no action takes place.

- standard DOS errors (Invalid handle, Invalid function number, etc...);

- the length of the buffer passed (register CX) is not correct (extended error 24, Bad request structure length);

- the command code is invalid (extended error 31, General failure);

- already 32 NAF are declared **and** the NAF to be declared is not already declared (extended error 29, Write fault).

7.1.1.4.2 Extraction action

The NAF uses the PCIDD$ Device Driver to extract itself, issuing an IOCTL write command, passing a structure containing the action code (Extract) and the handle of the NAF.

The following operation will take place in order:

- open the 'PCIDD$' driver;
- prepare the following structure:
  - one word: command code, 0 x 5845 (characters 'EX', EXtraction);
  - one double-word: address of the NAF entry point.
- issue a IOCTL system call write command:
  - CX contains the size of the extraction structure (6);
  - DS:DX point to the structure.
- check the success of the operation (check CARRY FLAG);
- in case of error, issue a Get Extended Error function call to get a more comprehensive error code;
- close the driver.

The command will be successful even if the NAF has not already been declared. In this case, no action takes place.

The command gives an error on the following cases. In these cases, no action takes place.

- standard DOS errors (Invalid handle, Invalid function number, etc...);
- the length of the buffer passed (register CX) is not correct (extended error 24, Bad request structure length);
- the command code is invalid (extended error 31, General failure).

7.1.2 MS-DOS for Profile B

As MS-DOS does not provide any multitasking facilities, Profile B is incorporated into the system as a background driver (terminate and stay resident). The interface between the application and Profile B is implemented by way of a software interrupt. The vector used for this shall be configurable both in Profile B and in the application. The default value for the software interrupt is 241 (0xF1). If another value is to be used, it can be specified as a parameter when Profile B is installed.

The functions described below are defined by appropriate register assignments in this software interrupt interface. The return values and parameter are normally supplied in register AX and ES:BX. Registers AX, BX, CX, DX and ES can be modified, other registers are retained. Profile B is allowed to enable interrupts during processing of these functions.

Profile B requires a maximum stack area of 512 bytes for the execution of all the functions incorporated. This stack area shall be made available by the application program. During processing the software interrupt Profile B may enable and/or disable interrupts.

The software interrupt for Profile B is defined according to the BIOS interrupt chaining structure.

API	**PROC**	**FAR**	**; ISDN-API interrupt service**
	JMP	**SHORT doit**	**; jump to start of routine**
	DD	**?**	**; chained interrupt**
	DW	**424BH**	**; interrupt chaining signature**
	DB	**80H**	**; first in chain flag**
	DW	**?**	**; reserved, should be 0**
	DB	**'CAPI'**	; Profile B **signature**
	DB	**'20'**	**; Version number**
doit:			

The characters 'CAPI20' can be requested by the application to check the presence of Profile B.

The pointer stipulated in messages DATA_B3_REQ and DATA_B3_IND is implemented as a FAR pointer under MS-DOS.

Memory layout is according to MS-DOS.

7.1.2.1 Message operations

7.1.2.1.1 CAPI_REGISTER

Description
This is the function the application uses to report its presence to Profile B. In doing so, the application provides Profile B with a memory area. A FAR pointer to this memory area is transferred in registers *ES:BX*. The size of the memory area is calculated according to the following formula:

CX + (DX * SI * DI)

The size of the message buffer used to store messages is transferred to the *CX* register. Choosing too small a value will result in messages being lost. A 'normal' application should calculate the necessary amount of memory according to following formula:

CX = 1 024 + (1 024 * DX)

In the *DX* register the application indicates the maximum number of logical connections opened simultaneously. An attempt to open more logical connections than stipulated here can be acknowledged with an error message from Profile B.

In the *SI* register the application sets the maximum number of received B3 data blocks that can be reported to the application simultaneously. The number of simultaneously available B3 data blocks has a decisive effect on the throughput of B3 data in the system and should be between 2 and 7. There shall be room for two B3 data blocks at least.

In the *DI* register the application sets the maximum size of the application data to be transmitted and received, that is the maximum *data length* parameter in messages **DATA_B3_REQ** and **DATA_B3_IND**. The default value for the protocol ISO 7776 [4] (X.75) is 128 octets. Profile B shall be able to support at least up to 2 048 octets, if an application sets register *DI* with corresponding values.

The application number is supplied in the *AX* register. In the event of an error, the *AX* register is returned with the value 0. The cause of the error is held in the *BX* register in this case.

CAPI_REGISTER	**0x01**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function code 0x01
ES:BX	FAR pointer to a memory block provided by the application. This memory area can (but need not) be used by Profile B to manage the message queue of the application. In addition, Profile B can (but also need not) provide the received data in this memory area.
CX	Size of message buffer
DX	Maximum number of level 3 connections
SI	Number of B3 data blocks available simultaneously
DI	Maximum size of a B3 data block

Return Value

Return	Value	Comment
AX	<> 0	Application number (ApplID)
	0x0000	Registration error, cause of error in BX register
BX		if AX == 0, coded as described in parameter Info class 0x10xx

NOTE: If the application intends to open a maximum of one layer 3 connection simultaneously and the standard protocols are used, the following register assignment is recommended:

CX = 2048, DX = 1, SI = 7, DI = 128

The resulting memory requirement is 2 944 bytes.

7.1.2.1.2 CAPI_RELEASE

Description
The application uses this function to log off from Profile B. The memory area indicated in the **CAPI_REGISTER** is released. The application is identified by the application number in the *DX* register. Any errors that occur are returned in register *AX*.

CAPI_RELEASE	**0x02**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x02
DX	Application number

Return Value

Return	Value	Comment
AX	0x0000	no error
	<> 0	Registration error, coded as described in parameter Info class 0x11xx

7.1.2.1.3 CAPI_PUT_MESSAGE

Description
With this function the application transfers a message to Profile B. A FAR pointer is transferred to the message in the *ES:BX* registers. The application is identified via application number in the *DX* register. Any errors that occur are returned in register *AX*.

CAPI_PUT_MESSAGE	**0x03**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x03
ES:BX	FAR pointer to the message
DX	Application number

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
NOTE: After **CAPI_PUT_MESSAGE** the application can use the memory area of the message again. The message shall not be modified by Profile B.		

7.1.2.1.4 CAPI_GET_MESSAGE

Description

With this function the application retrieves a message from Profile B. The application can only retrieve those messages intended for the stipulated application number. A FAR pointer is set to the message in the *ES:BX* registers. If there is no message for the application, the function returns immediately. Register *AX* contains the corresponding error value. The application is identified via the application number in the *DX* register. Any errors that occur are returned in register *AX*.

CAPI_GET_MESSAGE	**0x04**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x04
DX	Application number

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
ES:BX		FAR pointer to message, if available
NOTE: The message may be invalidated the next time **CAPI_GET_MESSAGE** is called.		

7.1.2.2 Other functions

7.1.2.2.1 CAPI_SET_SIGNAL

Description
The application can use this function to activate usage of the interrupt call-back function. A FAR pointer to an interrupt call-back function is specified in the *ES:BX* registers. The signalling function can be deactivated by a **CAPI_SET_SIGNAL** with register assignment *ES:BX* = 0000:0000. The application is identified via the application number in the DX register. Any errors that occurred are returned in the *AX* register.

CAPI_SET_SIGNAL	**0x05**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x05
DX	Application number
SI:DI	Parameter passed to call-back function
ES:BX	FAR pointer to call-back function

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

NOTE: The call-back function is called as an interrupt by Profile B**,** after;
- any message is queued in application's message queue;
- a notified busy condition is cleared;
- a notified queue full condition is cleared.

Interrupts are disabled. The call-back function shall be terminated via IRET. All registers shall be preserved. At the time of calling, at least 32 bytes are available on the stack.

The call-back function shall be called with interrupts disabled. Profile B shall not call this function recursively, even if the call-back function enables interrupts. Instead, the call-back function shall be called again after returning to Profile B.

The call-back function is allowed to use Profile B operations **CAPI_PUT_MESSAGE**, **CAPI_GET_MESSAGE**, and **CAPI_SET_SIGNAL**. In that case the application shall be aware that interrupts may be enabled by Profile B.

In case of local confirmations (e.g. LISTEN_CONF) the call-back function may be activated before the operation CAPI_PUT_MESSAGE returns to the application.

Parameter DX, SI and DI shall be passed to the call-back function with the same values of the corresponding parameters to **CAPI_SET_SIGNAL**.

7.1.2.2.2 CAPI_GET_MANUFACTURER

Description
With this function the application determines the manufacturer identification of Profile B**.** In registers *ES:BX* a FAR pointer is transferred to a data area of 64 bytes. The manufacturer identification, coded as a zero terminated ASCII string, is present in this data area after the function has been executed.

CAPI_GET_MANUFACTURER	**0xF0**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF0
ES:BX	FAR pointer to buffer

Return Value

Return	Comment
ES:BX	buffer contains manufacturer identification with ASCII coding. The end of the identification is indicated with a 0 byte.

7.1.2.2.3 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

CAPI_GET_VERSION	**0xF1**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF1

Return Value

Return	Comment
AH	Profile B major version: 2
AL	Profile B minor version: 0
DH	Manufacturer specific major number
DL	Manufacturer specific minor number

7.1.2.2.4 CAPI_GET_SERIAL_NUMBER

Description

With this function the application determines the (optional) serial number of Profile B**.** In registers *ES:BX* a FAR pointer to a data area of 8 bytes is transferred. The serial number, coded as a zero terminated ASCII string, is present in this data area in the form of a seven-digit number after the function has been executed. If no serial number is supplied, the serial number is an empty string.

CAPI_GET_SERIAL_NUMBER	**0xF2**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF2
ES:BX	FAR pointer to buffer

Return Value

Return	Comment
ES:BX	The (optional) serial number is read in plain text in the form of a 7-digit number. If no serial number is to be used, a 0 byte shall be written at the first position in the buffer. The end of the serial number is indicated with a 0 byte.

7.1.2.2.5 CAPI_GET_PROFILE

Description

The application uses this function to get the capabilities from Profile B. Registers *ES:BX* contain a FAR pointer to a data area of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. Register *CX* contains the controller number (bit 0..6) for which this information is requested.

CAPI_GET_PROFILE	**0xF3**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Functional Code 0xF3
CX	controller number (if 0, only number of controllers is returned)
ES:BX	FAR pointer to buffer

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
2 octets	number of installed controllers, least significant octet first
2 octets	number of supported B-channels, least significant octet first
4 octets	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4..31: reserved
4 octets	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for fax group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
4 octets	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
4 octets	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 [16] Appendix II. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
24 octets	reserved for Profile B usage
20 octets	manufacturer specific information
NOTE: This function can be extended, so an application has to ignore unknown bits. Profile B shall set every reserved field to 0.	

7.1.2.2.6 CAPI_MANUFACTURER

Description
This function is manufacturer specific.

CAPI_MANUFACTURER	0xFF

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xFF
Manufacturer specific	

Return Value

Return	Comment
Manufacturer specific	

7.2 Windows version 3.x

7.2.1 Windows operating system specific implementation for Profile A

7.2.1.1 Introduction

Except for the PciGetHandles function call, the DLL mechanism is the basic mechanism used to support the Profile A exchange method under Windows. Every NAF have to be DLL and have to export an entry point per Profile A function using the same name (PciGetProperty, PciRegister, PciGetMessage, PciPutMessage, PciSetSignal, PciDeregister).

NOTE: Function names exported by the NAF are the same as the description made in the subclause 5.3 but the parameters are different.

PciGetHandles needs an access to the PCI.INI file.

PciRegister and PciGetProperty check if the DLL, accessible by its name, is available.

To access a NAF the only need for a PUF is to know the name of the DLL. The address access to the DLL may be provide transparently to the PUF inside the Pci's exchange mechanism functions as shown in the annex J.

The PciRegister function dynamically loads the NAF. It needs to keep trace of the handle of the NAF as a DLL, so this handle is part of the Exchange Identifier. The NAF need also to keep trace of the PUF, so it assigns an Identifier to the PUF at registration time. This NAF-provided Identifier is the other part of the Exchange Identifier.

Under Windows, the common calling conventions to provide parameter to a DLL is the PASCAL calling convention. This convention is also used by the Profile A exchange method in that case.

Pointer parameters are far. The PCIMPB structure is always passed via a pointer. The Exchange Identifier structure is also always passed via a pointer.

The structure alignment is **byte**.

7.2.1.2 Implementation of basic type

Under Windows, the following values shall be used:

PCI_HANDLE	name of the DLL
PCI_EXID	Structure contents handle provided by Windows when the DLL is loaded (hInstance) Unique Identifier provided by NAF to identify the PUF
PCI_PROCEDURE	exported function address (FARPROC) provided by the PUF
PCI_INTEGER	2 bytes
PCI_BYTEARRAY	far pointer

7.2.1.3 C structures and function prototypes

```
/* Basic types */
typedef SHORT     PCI_INTEGER;
typedef LPSTR     PCI_BYTEARRAY;
typedef LPSTR     PCI_HANDLE;
typedef struct    {
        HINSTANCE       DLLInstance;
        PCI_INTEGER     Exchange_Id;
        } PCI_EXID;
typedef void (far pascal *PCI_PROCEDURE)(void);

/*
 * Structures
 */
struct pci_mpb {
                PCI_INTEGER     MessageID;
                PCI_INTEGER     MessageMaximumSize;
                PCI_INTEGER     MessageActualUsedSize;
                PCI_INTEGER     DataMaximumSize;
                PCI_INTEGER     DataActualUsedSize;
};

struct pci_register {                   /* structure containing registering info */
        PCI_INTEGER PUFVersion;             /* optional: give PUF version */
        PCI_INTEGER PUFType;                /* optional: give PUF type */
        PCI_INTEGER MaxMsgSize;             /* return: max size of a message */
};

struct pci_opsys {                  /* structure containing specific operating system info */
        int     DummyParameter;         /* No specific requirement for WINDOWS */
};

/* Exchange functions prototypes */

PCI_INTEGER far PASCAL PciGetHandles (      PCI_INTEGER MaxHandles,
                                    PCI_BYTEARRAY PCIHandles,
                                    PCI_INTEGER far * ActualHandles);

PCI_INTEGER far PASCAL PciGetProperty (     PCI_HANDLE PCIHandle,
                                    PCI_INTEGER MaximumSize,
                                    PCI_BYTEARRAY Property,
                                    PCI_INTEGER far * ActualSize);

PCI_INTEGER far PASCAL PciRegister (    PCI_HANDLE PCIHandle,
                                    struct pci_register * PCIRegisterInfo,
                                    struct pci_opsys * PCIOpSysInfo,
                                    PCI_EXID far *ExID);

PCI_INTEGER far PASCAL PciDeregister (  PCI_EXID far *ExID);
```

```
PCI_INTEGER far PASCAL PciPutMessage (      PCI_EXID far *ExID,
                                    struct pci_mpb far *PCIMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data);

PCI_INTEGER far PASCAL PciGetMessage (      PCI_EXID far *ExID,
                                    struct pci_mpb far *PCIMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data);

PCI_INTEGER far PASCAL PciSetSignal (   PCI_EXID far *ExID,
                                    PCI_INTEGER Signal,
                                    PCI_PROCEDURE SignalProcedure);
```

7.2.1.4 Description of functions

This subclause describes the implementation, under Windows, of the Profile A exchange method functions. During a PUF to NAF call, the size of the stack shall be at least 1 024 bytes deep.

7.2.1.4.1 PciGetHandles

Under WINDOWS, the PciGetHandles uses a PCI.INI file in the WINDOWS directory to get available PCI_HANDLEs.

The section [Drivers] in the PCI.INI file contains all entries of installed NAFs. Each entry has the format:

pciDriver<number>=DLLName (number=1..32)

The following operations shall get all names of installed NAF drivers:

- loops from 1 to 32
  - constructs of the keyName 'pciDriver' associated to the current loop value;
  - issue a GetPrivateProfileString using:
    sectionKey = 'DRIVERS',
    the keyName constructs before,
    no default value,
    a maximum size equal to 128,
    FileName = 'PCI.INI'.

7.2.1.4.2 PciGetProperty

This function is in charge of providing the PUF with the PROPERTY of the NAF. Implicitly it checks if the NAF is available, when loading the library via the LoadLibrary function.

The following operations shall take place, in order:

- load the DLL;
- get the address of the PciGetProperty function exported by the NAF;
- call to this address with the parameters provided by the PUF;
- free the loaded library.

7.2.1.4.3 PciRegister

This function is in charge to provide an association between a PUF and a NAF. The NAF is loaded and the DLLInstance part of the Exchange Identifier is provided. The availability of the chosen NAF is checked during the load of the library. The library is identified by its name. Parameters for the registration operation are brought together a structure:

- PUFType (PCI_INTEGER);
- PUFVersion (PCI_INTERGER);
- MaxMsgSize (PCI_INTEGER) where the NAF will give the maximum size for a message.

The following operations shall take place, in order:

- load the DLL;

- provide the DLLInstance part of the Exchange Identifier with the DLL Instance;

- get the address of the PciRegister function exported by the NAF;

- call to this address to inform the NAF of a new PUF. The address of the registration parameters structure and the address of the Exchange Identifier structure are passed to the NAF as parameters;

- on return from the NAF, the Exchange_Id part of the Exchange Identifier and the maximum message size parameter of the registration parameter structure have been provided by the NAF;

- return to the PUF with the return code from the NAF.

7.2.1.4.4 PciDeregister

This function is in charge to disassociate a PUF and a NAF. The DLL usage number shall be decremented by Windows but the DLL is not freed from the memory each time a PUF deregisters a NAF.

The following operations shall take place, in order:

- get the address of the PciDeregister function exported by the NAF;

- call to this address to inform the NAF of the end of the association. The PCI_EXID is passed to the NAF by address;

- free the DLL.

7.2.1.4.5 PciPutMessage

This function is in charge to provide a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as in the description of the PciGetMessage in subclause 5.3.

The following operations shall take place, in order:

- get the address of the PciPutMessage function exported by the NAF;
- call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.2.1.4.6 PciGetMessage

This function is in charge to provide a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as in the description of the PciGetMessage in subclause 5.3. Buffers provided by the PUF are directly used by the NAF.

The following operations shall take place, in order:

- Get the address of the PciGetMessage function exported by the NAF;
- Call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.2.1.4.7 PciSetSignal

This function allows a PUF to provide a direct information mechanism to be used by the NAF in case of incoming event. Two mutually exclusive mechanisms are offered under Windows:

- A signal procedure mechanism;
- A user message mechanism.

Once a mechanism is chosen by the PUF, the other is deactivated by the NAF for that particular PUF. Both mechanisms have to be supported by a NAF.

The first mechanism does not used the Signal parameter. This parameter shall be set to 0.

The second mechanism used the Signal parameter to identify the value associated with the WM_USER WINDOWS message. In that case, the Signal parameter shall not be equal to 0.

7.2.1.4.7.1 Signal mechanism procedure

The routine address, provided by the PUF in the SignalProcedure parameter, is used directly by the NAF. It shall be made accessible to the NAF before it is provided by the PUF. The routine is called without any parameters.

In that case, the Signal parameter is not used but the parameter shall be passed to the NAF with the 0 value.

The stack used during the call to the SignalProcedure is not the PUF's one. The SignalProcedure shall be compiled without assuming SS equal to DS, i.e. as a DLL.

The NAF is allowed to call the PUF to reissue a signal call. To avoid big stack requirement, the NAF shall wait the return from the PUF signal procedure before re-issuing the next signal call.

The PUF call back to the NAF during the signal procedure treatment is not allowed. The stack size is not guaranty when the NAF calls the PUF. Consequently, the stack requirements for the PUF treatment shall be as small as possible.

7.2.1.4.7.2 User message mechanism procedure

The Signal parameter contains a PUF value to be added to the WM_USER WINDOWS message constant. This message is sent to a PUF Window. The HANDLE for this Window is provided by the PUF in the low word of the SignalProcedure parameter of the PciSetSignal function. It shall be a valid HANDLE WINDOW (HWND).

When the NAF issues the WM_USER + Signal message to the PUF, it uses a WINDOWS API PostMessage call. The PUF will find as third parameter (known as wParam) the type of the message received. In the fourth parameter (lParam), the PUF will find, as high word, the size of the Message associated to this message and as low word, the size of the Data associated. The call will look like:

```
PostMessage     (LOWORD(SignalProcedure),
                 WM_USER+Signal,
                 MessageID,
                 (DWORD) (MessageSize << 16) | (DataSize));
```

As the PostMessage WINDOWS API is used, the PUF is allowed to call back the NAF during the message treatment.

This mechanism is simple to be implemented but an important constraint shall be pointed out:

- Under WINDOWS, a PostMessage call can fail due to a lack of room available in the message queue. The PUF is in charge to treat fast enough messages to insure that no NAF message will be lost. The PUF cannot rely on a failed message to be reissued by the NAF.

7.2.1.4.7.3 Deactivation mechanism

To deactivate any signal mechanism the PciSetSignal function Signal and SignalProcedure parameters shall be set to NULL. Once deactivated, the previous mechanism shall no longer be used by the NAF to call the PUF.

7.2.1.5 Availability of NAF's PCI_HANDLE

To be accessible via the PciGetHandles function call, a NAF shall issue a declaration action. The inverse action - extraction from the list of available NAFs - is described too. These actions are operating system specific.

7.2.1.5.1 Declaration action

First, the NAF may get the list of available PCI_HANDLEs to check if not already declared. The mechanism the NAF uses is the same as any PUF to get available NAF: PciGetHandles (see subclause 7.2.1.4.1).

If not yet declared, the NAF includes its own PCI_HANDLE into the list.

```
PCI_BYTEARRAY ownDLLName = "xxx";
PCI_BYTE      driverName[128];
WORD                index;
char          keyName[20];

/* Check if NAF not already installed */
for (index = 1; index <=  32; index++)
        {
        sprintf(keyName, "pciDriver%d", index);
        if (GetPrivateProfileString(    "DRIVERS", /* Section name */
                        keyName,    /* "pciDriver"+1..n */
                        NULL,               /* No default needed */
                        driverName,
                        sizeof(driverName),
                        "PCI.INI") > 0)
                {
                if (strcmpi(driverName, ownDLLName) == 0) return; /* NAFinstalled, OK return */
                }
        }
/* Search a free pciDriver position */
for (index = 1, index <= 32; index++)
        {
        sprintf(keyName, "pciDriver%d", index);
        if (GetPrivateProfileString(    "DRIVERS", /* Section name */
                        keyName,    /* "pciDriver"+1..n */
                        NULL,               /* No default needed */
                        driverName,
                        sizeof(driverName),
                        "PCI.INI") == 0)
                {
                /* Entry does not exist, add own NAF Driver name */
                WritePrivateProfileString("DRIVERS", keyName, ownDLLName, "PCI.INI");
                return;
                }
        }
```

The maximum number of NAF than can be registered is 32.

7.2.1.5.2 Extraction action

First, the NAF gets the list of available PCI_HANDLEs to check if it is declared. If so, the NAF removes its own PCI_HANDLE from the driver list in "PCI.INI".

```
PCI_BYTEARRAY ownDLLName = "xxx";
PCI_BYTE      driverName[128];
WORD                index;
char          keyName[20];
```

```
for (index = 1, index <= 32; index++)
        {
        sprintf(keyName, "pciDriver%d", index);
        if (GetPrivateProfileString(        "DRIVERS", /* Section name */
                                keyName,    /* "pciDriver"+1..n
                                NULL,               /* No default needed */
                                driverName,
                                sizeof(driverName),
                                "PCI.INI") > 0)
                {
                /* Check for own name */
                if (strcmpi(driverName, ownDLLName) == 0)
                        {
                        /* Remove the name of the Driver */
                        WritePrivateProfileString("DRIVERS", keyName, "", "PCI.INI");
                        }
                }
        }
```

7.2.2 Windows (application level) for Profile B

In a PC environment with the MS-DOS extension Windows an application can access Profile B services via a DLL (Dynamic Link Library). The interface between applications and Profile B is realised as a function interface. An application can issue Profile B function calls to perform Profile B operations.

The DLL providing the function interface shall be named "CAPI20.DLL". All functions exported by this library shall be called with a FAR call according to the PASCAL calling convention. This means all parameters are pushed on the stack (first parameter is pushed first), the called function shall clear up the stack before it returns to the caller.

The functions are exported under following names and ordinal numbers:

CAPI_MANUFACTURER (reserved)	CAPI20.99
CAPI_REGISTER	CAPI20.1
CAPI_RELEASE	CAPI20.2
CAPI_PUT_MESSAGE	CAPI20.3
CAPI_GET_MESSAGE	CAPI20.4
CAPI_SET_SIGNAL	CAPI20.5
CAPI_GET_MANUFACTURER	CAPI20.6
CAPI_GET_VERSION	CAPI20.7
CAPI_GET_SERIAL_NUMBER	CAPI20.8
CAPI_GET_PROFILE	CAPI20.9
CAPI_INSTALLED	CAPI20.10

These functions can be called by an application according to the DLL conventions as imported functions. If an application calls any function of the DLL with whatever function it shall ensure that there are at least 512 bytes left on the stack.

All pointers that are passed from the application program to Profile B, or vice versa, in function calls or in messages are 16:16 segmented protected mode pointers. This especially applies to the data pointer in **DATA_B3_REQ** and **DATA_B3_IND** messages.

In the Windows 3.x environment following types are used to define the functional interface:

WORD	16 bit unsigned integer
DWORD	32 bit unsigned integer
LPVOID	16:16 (segmented) protected mode pointer to any memory location
LPVOID *	16:16 (segmented) protected mode pointer to a LPVOID
LPBYTE	16:16 (segmented) protected mode pointer to a character string
LPWORD	16:16 (segmented) protected mode pointer to a 16 bit unsigned integer value
CAPIENTRY	WORD FAR PASCAL (according to Windows DLL calling convention)

7.2.2.1 Message operations

7.2.2.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the four parameters MessageBufferSize, maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs.

For a 'normal' application the size of the message buffer should be calculated using following formula:

MessageBufferSize = 1024 + (1024 * maxLogicalConnection)

Function call

```
CAPIENTRY CAPI_REGISTER (   WORD MessageBufferSize,
                            WORD maxLogicalConnection,
                            WORD maxBDataBlocks,
                            WORD maxBDataLen,
                            LPWORD pApplID);
```

Parameter	Comment
MessageBufferSize	Size of Message Buffer
maxLogicalConnection	Maximum number of logical connections
maxBDataBlocks	Number of data blocks available simultaneously
maxBDataLen	Maximum size of a data block
pApplID	Pointer to the location where Profile B should place the assigned application identification number

Return Value

Return Value	Comment
0x0000	Registration successful - application identification number has been assigned
All other values	Coded as described in parameter info class 0x10xx

7.2.2.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. Profile B shall release all resources that have been allocated for the application.

The application is identified by the application identification number that had been assigned in the previous CAPI_REGISTER operation.

Function call

CAPIENTRY CAPI_RELEASE (WORD ApplID);

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER

Return Value

Return Value	Comment
0x0000	Release of the application successful
All other values	Coded as described in parameter info class 0x11xx

7.2.2.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an application identification number.

Function call

CAPIENTRY CAPI_PUT_MESSAGE(WORD ApplID, **LPVOID pCAPIMessage);**

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
pCAPIMessage	16:16 (segmented) protected mode pointer to the message that is passed to Profile B

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx
NOTE: When the process returns from the function call the message memory area can be reused by the application.	

7.2.2.1.4 CAPI_GET_MESSAGE

Description

With this operation the application retrieves a message from Profile B. The application can only retrieve those messages intended for the stipulated application identification number. If there is no message waiting for retrieval, the function returns immediately with an error code.

Function call

```
CAPIENTRY CAPI_GET_MESSAGE (    WORD ApplID,
                                LPVOID *ppCAPIMessage);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
ppCAPIMessage	16:16 (segmented) protected mode pointer to the memory location where Profile B should place the 16:16 (segmented) protected mode pointer to the retrieved message

Return Value

Return Value	Comment
0x0000	Successful - Message was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx

NOTE: The received message may become invalid the next time the application issues a CAPI_GET_MESSAGE operation for the same application identification number. This especially matters in multi threaded applications where more than one thread may execute CAPI_GET_MESSAGE operations. The synchronisation between threads shall be done by the application.

7.2.2.2 Other functions

7.2.2.2.1 CAPI_ SET_SIGNAL

Description
This operation is used by the application to install a mechanism which signals the application the availability of a message or the clearing of an internal busy/queue full condition. All restrictions of interrupt context will apply to the call-back function.

Function call

```
CAPIENTRY CAPI_SET_SIGNAL (     WORD ApplID,
                                VOID (FAR PASCAL *CAPI_Callback)
                                (WORD ApplID, DWORD Param),
                                DWORD Param
                                );
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
CAPI_Callback	address of the call-back function. The function can called in an interrupt context (see note). Value 0x00000000 disables the call-back notification.
Param	additional parameter of call-back function

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

NOTE: The notification takes place**,** after:
- any message is queued in application's message queue;
- a notified busy condition is cleared;
- a notified queue full condition is cleared.

In case of local confirmations (e.g. LISTEN_CONF) the notification may be activated before the operation CAPI_PUT_MESSAGE returns to the application.
The call-back function shall be called using following conventions:

```
VOID FAR PASCAL CAPI_Callback (
        WORD ApplID,
        DWORD Param
);
```

Data segment register DS is undefined (use MakeProcInstance() or _setds). A stack of at least 512 bytes is set up by Profile B.
The call-back function may be called at interrupt context (i.e., every data and code accessed by the call-back function has to be prevented from being paged out by Windows' VMM, e.g. by using fixed segments in its own DLL and/or by applying GlobalPageLock() to used selectors).
PostMessage() and PostAppMessage() are the only windows API functions which can be called.
CAPI_PUT_MESSAGE, CAPI_GET_MESSAGE and CAPI_SET_SIGNAL are the only Profile B functions which can be called.
The call-back function shall not be re-entered by Profile B. Instead it shall be called again after returning, if a new event has occurred during processing.

7.2.2.2.2 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B (DLL). SzBuffer on call is a 16:16 (segmented) protected mode pointer to a buffer of 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

Function call

```
CAPIENTRY CAPI_GET_MANUFACTURER (LPBYTE SzBuffer);
```

Parameter	Comment
SzBuffer	16:16 (segmented) protected mode pointer to a buffer of 64 bytes

Return Value

Return Value	Comment
0x0000	No error

7.2.2.2.3 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

Function call

```
CAPIENTRY CAPI_GET_VERSION (    LPWORD pCAPIMajor,
                                LPWORD pCAPIMinor,
                                LPWORD pManufacturerMajor,
                                LPWORD pManufacturerMinor);
```

Parameter	Comment
pCAPIMajor	16:16 (segmented) protected mode pointer to a WORD receiving Profile B major version number: 2
pCAPIMinor	16:16 (segmented) protected mode pointer to a WORD receiving Profile B minor version number: 0
pManufacturerMajor	16:16 (segmented) protected mode pointer to a WORD receiving manufacturer specific major number
pManufacturerMinor	16:16 (segmented) protected mode pointer to a WORD receiving manufacturer specific minor number

Return Value

Return	Comment
0x0000	No error, version numbers are copied

7.2.2.2.4 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. SzBuffer on call is a 16:16 (segmented) protected mode pointer to a string buffer of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

Function call

```
CAPIENTRY CAPI_GET_SERIAL_NUMBER (LPBYTE SzBuffer);
```

Parameter	Comment
SzBuffer	16:16 (segmented) protected mode pointer to a buffer of 8 bytes

Return Value

Return	Comment
0x0000	No error SzBuffer contains the serial number in plain text in the form of a 7-digit number. If no serial number is provided by the manufacturer, an empty string is returned.

7.2.2.2.5 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. SzBuffer on call is a 16:16 (segmented) protected mode pointer to a buffer of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. *CtrlNr* contains the controller number (bit 0..6), for which this information is requested.

```
CAPIENTRY CAPI_GET_PROFILE (    LPBYTE SzBuffer,
                                WORD CtrlNr
                                );
```

Parameter	Comment
SzBuffer	16:16 (segmented) protected mode pointer to a buffer of 64 bytes
CtrlNr	Number of Controller. If 0, only number of installed controllers is given to the application.

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
WORD	number of installed controllers, least significant octet first
WORD	number of supported B-channels, least significant octet first
DWORD	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4..31: reserved
DWORD	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
DWORD	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
DWORD	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
6 DWORDs	reserved for Profile B usage
5 DWORDs	manufacturer specific information
NOTE: This function can be extended, so an application shall ignore unknown bits. Profile B shall set every reserved field to 0.	

7.2.2.2.6 CAPI_INSTALLED

Description
This function can be used by an application to determine if the ISDN hardware and necessary drivers are installed.

Function call

CAPIENTRY CAPI_INSTALLED (void)

Return Value

Return	Comment
0x0000	Profile B is installed
any other value	Coded as described in parameter info class 0x10xx

7.3 UNIX

7.3.1 UNIX Operating System specific implementation for Profile A

7.3.1.1 Introduction

The PCI exchange functions described in the subclause 5.3 have to be mapped to appropriate functions supplied by the UNIX STREAMS kernel mechanism.

The binary compatible interface to a NAF running under the UNIX operating system shall be implemented using the STREAMS kernel mechanism.

Descriptions was made using "C" language because it is the natural language in the UNIX environment.

7.3.1.2 Implementation of basic types

The mapping of the basic types of the exchange method to "C" language types are defined as the following:

Basis type	Mapping and usage
PCI_INTEGER	Can be implemented as 2 or 4 bytes signed integer, whatever is defined within the underlying UNIX system as system constant for the 'int' type.
PCI_BYTEARRAY	Implemented as pointer to 'char' type.
PCI_EXID	Implemented as 'int' type. Since the exchange method is implemented using STREAMS, the Exchange-ID has the same value and type as the UNIX file descriptor provided by the STREAMS kernel mechanism.
PCI_HANDLE	Implemented as pointer to 'char' type, e.g. a UNIX character-string. The string shall to contain the name of the STREAMS device the NAF is implemented in.
PCI_PROCEDURE	Implemented as address of a function returning 'void' type, as defined by UNIX signal() system call.

7.3.1.3 Parameter passing conventions

For parameter passing the usual "C" conventions are applying:

- call values are either passed by value (e.g. PCI_INTEGER, PCI_EXID), or by the use of a pointer (e.g. PCI_BYTEARRAY, PCI_HANDLE);

- return values are passed by giving a pointer for filling in the value (passing by reference).

Errors will occur inside the NAF driver are returned as positive integers (PCI_INTEGER). Their values are defined in the clause 5. As defined there, a value of 0 stands for "no error" (Success).

Errors occurred inside of the PCI exchange functions should be returned as negative integers (PCI_INTEGER). Their values are not defined. They are NAF implementation dependent.

7.3.1.4 Definition of types, constants and function-prototypes

If alignment is necessary on the UNIX target system, the size of the **int** type is employed.

```
/*
 * Basic types
 */

typedef int          PCI_INTEGER;
typedef char *       PCI_BYTEARRAY;
typedef int          PCI_EXID;
typedef char *       PCI_HANDLE;
typedef void (*      PCI_PROCEDURE) ();


/*
 * Structures
 */

struct pci_mpb {
        PCI_INTEGER     MessageID;
        PCI_INTEGER     MessageMaximumSize;
        PCI_INTEGER     MessageActualUsedSize;
        PCI_INTEGER     DataMaximumSize;
        PCI_INTEGER     DataActualUsedSize;
};


/*
 * Exchange functions prototypes
 */
PCI_INTEGER PciGetHandles (PCI_INTEGER      MaxHandles,
                           PCI_BYTEARRAY    PCIHandles,
                           PCI_INTEGER *    ActualHandles);

PCI_INTEGER PciGetProperty (PCI_HANDLE          PCIHandle,
                           PCI_INTEGER          MaximumSize,
                           PCI_BYTEARRAY        Property,
                           PCI_INTEGER *        ActualSize);

PCI_INTEGER PciRegister (   PCI_HANDLE          PCIHandle,
                           PCI_INTEGER          PUFVersion,
                           PCI_INTEGER          PUFType,
                           PCI_EXID *       ExID,
                           PCI_INTEGER *    MaxMsgSize);

PCI_INTEGER PciDeregister (  PCI_EXID        ExID);
```

```
PCI_INTEGER PciPutMessage (     PCI_EXID            ExID,
                        struct pci_mpb *    PCIMPB,
                        PCI_BYTEARRAY Message,
                        PCI_BYTEARRAY Data);

PCI_INTEGER PciGetMessage (     PCI_EXID            ExID,
                        struct pci_mpb *    PCIMPB,
                        PCI_BYTEARRAY Message,
                        PCI_BYTEARRAY Data);

PCI_INTEGER PciSetSignal (      PCI_EXID    ExID,
                        PCI_INTEGER         Signal,
                        PCI_PROCEDURE       SignalProcedure);
```

7.3.1.5 Adaptation to the STREAMS kernel mechanism

7.3.1.5.1 General

A NAF implemented into the UNIX kernel shall oppose its Profile A interface via the STREAMS kernel mechanism. For each implemented NAF one STREAMS access shall be provided, independent of the amount of ISDN accesses the NAF provides. Such a STREAMS access can in principle, if implemented by the NAF, be used by several PUFs. Furthermore, as a consequence of the UNIX architecture, one PUF can access several STREAMS and thus several NAFs simultaneously. NAFs shall be defined as CLONE Streams.

The UNIX STREAMS kernel mechanism provides two queues, a write queue and a read queue. Information sent by the exchange functions to the stream driver (downstream information) are placed into the write queue by a STREAMS component called the stream head. Stimulating the stream head to do so is achieved by issuing the STREAMS putmsg() system call.

The stream driver can access the information of the write queue, processes it and places resulting information into the read queue. The content of the read queue (upstream information) is available to the exchange functions by use of the STREAMS getmsg() system call.

7.3.1.5.2 Communication between PUF exchange functions and NAF stream driver

The communication between an exchange function and the NAF stream driver is carried out by the exchange function by means of getmsg() or putmsg() in the case of PciGetMessage() and PciPutMessage(), and ioctl() in the case of all other functions.

The information transported through this stream is called a STREAMS message. STREAMS messages should not be confused with the messages defined in the Profile A.

The STREAMS mechanism divides the PCI message into two parts: a control part and a data part. For messages exchanged via PciGetMessage() and PciPutMessage(), the control part of the STREAMS message contains the PCI message and the data part will contain the data part of the PCI message. The NAF driver receives the lengths of the individual parts of the PCI message by means of the standard UNIX getmsg() and putmsg() mechanism.

For all other messages, the individual command is passed to the NAF driver in the ioc_cmd field of the struct iocblk structure. The data part associated with this command is passed to the NAF driver in the data blocks of the M_IOCTL message.

Definitions of terms:

mp	is of type mblkt_t * (see /usr/include/sys/stream.h)
struct iocblk	type defined in /usr/include/sys/stream.h

The NAF STREAMS driver can obtain the information necessary for its operation by using the following mechanisms:

1. PCI Messages exchanged via PciPutMessage()

Information	**Availability**
Length of control part	mp->b_wptr - mp->b_rptr
Contents of control part	mp->b_rptr
Presence of a data part	mp->b_cont != NULL
Length of data part	msdgsize(mp)
Contents of data part	mp->b_cont->b_rptr

2. PCI Messages exchanged via the ioctl() mechanism

Information	**Availability**
Requested function	((struct iocblk *)mp->b_rptr)->ioc_cmd
Length of control part	((struct iocblk *)mp->b_rptr)->ioc_count
Contents of control part	mp->b_cont->b_rptr
Room for returned data	mp->b_cont->b_rptr
	((struct iocblk *)mp->b_rptr)->ioc_rval

The requested function shall be defined as follows:

```
#define PCI_PROPERTY        (('Z' << 8) | 1)
#define PCI_REGISTER        (('Z' << 8) | 2)
#define PCI_DEREGISTER          (('Z' << 8) | 3)
#define PCI_SETSIGNAL           (('Z' << 8) | 4)
```

7.3.1.5.3 Special considerations

Several NAF implementation aspects have to be considered by the PUF implementing the exchange functions:

- the PUF grants the NAF the permission to put incoming PCI messages on the read-side queue, thereby using this queue for buffering. Flow control is achieved by the standard UNIX highwater-lowwater mark mechanism which allows the NAF STREAMS driver to handle flow control transparently on the driver level;

- the size of a stream queue element is limited. A NAF stream driver shall be able to provide 4 096 bytes as data part of the stream message on the PUF's request, but it shall also guarantee this amount as the maximum delivered value. However, data block sizes of more than 4 096 bytes can be supported if the stream is put into "message non-discard mode" (see streamio(7)). Should a message with a data block size of more than 4 096 bytes arrive at the stream head, a call to PciGetMessage will return the first 4 096 bytes of the data block and successive calls to PciGetMessage will return the additional data blocks. Each of the additional calls to PciGetMessage will return a message whose control part length will be zero;

- only the UNIX SIGPOLL signal shall be issued by the NAF implementation.

7.3.1.6 Description of functions

This subclause describes the implementation of the PCI exchange functions using the UNIX STREAMS mechanism. The description of each function is divided into 3 parts:

1) Function body: function body description, including general description of the function behaviour
2) STREAMS putmsg(): Structure set-up for call to putmsg()
3) STREAMS getmsg(): Structure contents after return from getmsg()

The prototypes of putmsg () and getmsg () functions are:

```
int putmsg (fd, ctlptr, dataptr, flags)
        int fd;                 /* File descriptor                  */
        struct strbuf *ctlptr;  /* Control part of the message      */
        struct strbuf *dataptr;         /* Data part of the message         */
        int flags;              /* Message priority.                */
int getmsg (fd, ctlptr, dataptr, flags)
        int fd;                 /* File descriptor                  */
        struct strbuf *ctlptr;  /* Control part of the message      */
        struct strbuf *dataptr;         /* Data part of the message         */
        int *flags;             /* Message priority.                */
with
struct strbuf {
        int maxlen              /*Maximum buffer length             */
        int len                 /* Length of data                   */
        char *buf               /* Pointer to buffer                */
        }
```

Alternatively, for PCI exchange functions which use the ioctl() mechanism the description of each function is divided into two parts:

1) Function body: function body description, including general description of the function behaviour
2) ioctl(): Structure set-up for call to ioctl()

The prototype of ioctl () is:

```
int ioctl (fd, command, arg)
        int fd;         /* File descriptor                          */
        int command;            /* ioctl command as defined in streamio(7)  */
        char *arg;      /* command specific argument                */
```

Whenever command is I_STR arg should point to a structure of type strioctl, where strioctl is defined as:

```
struct strioctl{
        int ic_cmd;     /* User-defined command                     */
        int ic_timeout;         /* Timeout for command              */
        int ic_len;     /* Length of data part to follow    */
        char *ic_dp;    /* Command-specific arguments               */
        }
```

7.3.1.6.1 PciGetHandles

Function body:

```
PCI_INTEGER PciGetHandles (PCI_INTEGER MaxHandles,
                           PCI_BYTEARRAY PCIHandles,
                           PCI_INTEGER *ActualHandles)
{
...
}
```

MaxHandle contains the maximum number of PCI_HANDLE the PCIHandles parameter can receive. On return, ActualHandles, which is a pointer to an integer value, will contain the number of PCI_HANDLE copied into the PCIHandles parameter.

This function shall:

- examine the directory **/etc/pcidd** to get the number and the PCI_HANDLEs available;
- update the PCIHandles and the ActualHandles parameters;
- return appropriate error code.

7.3.1.6.2 PciGetProperty

Function body:

```
PCI_INTEGER PciGetProperty (    PCI_HANDLE          PCIHandle,
                                PCI_INTEGER         MaximumSize,
                                PCI_BYTEARRAY       NAFProperty,
                                PCI_INTEGER         *ActualSize)
{
        struct strioctl strioctl;
        extern int          errno;
        int             filedes;
}
```

PCIHandle points to the path name of the STREAMS device, MaximumSize is the size of the buffer to hold the properties. NAFProperty is the pointer to this buffer and ActualSize is a pointer to a integer value receiving the actual size of the property information in the NAF on return.

This function shall:

- open the STREAMS device using PCIHandle;
- issue the ioctl() call;
- retrieve the value of ActualSize and the error code;
- close the STREAMS device;
- return appropriate error code.

STREAMS ioctl():

The ic_cmd component shall be set to PCI_PROPERTY, the ic_len component shall be set to MaximumSize and the ic_dp component shall be set to point to the NAFProperty buffer.

Upon return from the ioctl() call the return value shall be checked against 0 which will indicate success. Any other return value indicates an error condition, which indicates that the errno variable contains the error condition. The ic_len component of the strioctl structure contains the number of bytes returned by the ioctl call. The ic_dp component points to the property returned.

NOTE: The size returned is always the size of the property inside the NAF.

```
strioctl.ic_cmd         = PCI_PROPERTY;
strioctl.ic_timout      = 0;
strioctl.ic_len         = MaximumSize;
strioctl.ic_dp          = (char *) NAFProperty;

if (ioctl (filedes, I_STR, &strioctl) == 0) {
        *ActualSize = strioctl.ic_len;
        return 0;
} else {
        *ActualSize = 0;
        return errno;
}
```

7.3.1.6.3 PciRegister

Function body:

```
PCI_INTEGER PciRegister      (PCI_HANDLE      PCIHandle,
                              PCI_INTEGER      PUFVersion,
                              PCI_INTEGER      PUFType,
                              PCI_EXID         *ExID,
                              PCI_INTEGER      *MaxMsgSize)
{
        struct strioctl       strioctl;
        struct pci_register_t pci_register;
        extern int            errno;
}
```

PCIHandle points to the path name of the STREAMS device. PUFVersion and PUFType are integers and set as indicated in the clause 6. ExID is a pointer to an integer receiving the returned Exchange-ID, which shall be equal to the UNIX file descriptor returned by the open() system call. MaxMsgSize is an integer receiving the message size of the NAF as described in the subclause 5.3.

This function shall:

- open the STREAMS device using PCIHandle;
- issue the ioctl() call;
- retrieve the return values from the pci_control structure;
- leave the STREAMS device open and assign file descriptor of open() call to ExID;
- return appropriate error code.

STREAMS ioctl():

The ic_cmd component shall be set to PCI_REGISTER, the ic_len component shall be set to the size of the pci_register structure and the ic_dp component shall be set to point to the pci_register structure which is set up with the values of PUFVersion and PUFType. Upon return from the ioctl() call the return value shall be checked against -1 which will indicate an error condition. The external variable errno will be set to indicate the specific error condition. Any other return value indicates success, and the return value of the ioctl call shall indicate the maximum PCI message size the NAF supports.

```
struct pci_register_t {
        int             puf_version;
        int             puf_type;
} pci_register;

pci_register.puf_version  = PUFVersion;
pci_register.puf_type     = PUFType;

strioctl.ic_cmd           = PCI_REGISTER;
strioctl.ic_timout        = 0;
strioctl.ic_len           = sizeof (pci_register);
strioctl.ic_dp            = (char *) &pci_register;

if ((*ExID = open (PCI_HANDLE, O_RDWR)) == -1) {
        *ExID = 0;
        return <cant_open_device: errno provides more information>;
}

if ((*MaxMsgSize = ioctl (*ExID, I_STR, &strioctl)) < 0) {
        *MaxMsgSize = 0;
        return errno;
}
else {
        return 0;
}
```

7.3.1.6.4 PciDeregister

Function body:

```
PCI_INTEGER PciDeregister    (PCI_EXID    *ExID)
{
        struct strioctlstrioctl;
        extern int              errno;
}
```

ExID identifies the open STREAMS device. It is identical to the file descriptor returned by the open() system call. This function shall:

- issue the ioctl() call;
- retrieve the error return code;
- close the STREAMS device;
- return appropriate error code.

STREAMS ioctl():

The ic_cmd component shall be set to PCI_DEREGISTER, the ic_len component shall be set to zero; the ic_dp component shall be set to NULL. Upon return from the ioctl() call the return value shall be checked against -1 which will indicate an error condition. The external variable errno will be set to indicate the specific error condition. Any other return value indicates success.

```
strioctl.ic_cmd      = PCI_DEREGISTER;
strioctl.ic_timout   = 0;
strioctl.ic_len      = 0;
strioctl.ic_dp       = (char *) NULL;


if (ioctl (*ExID, I_STR, &strioctl) == -1) {
        return errno;
}
else {
        close (*ExID);
        return 0;
}
```

7.3.1.6.5 PciPutMessage

Function body:

```
PCI_INTEGER PciPutMessage (PCI_EXID   ExID,
                           struct pci_mpb          *PCIMPB,
                           PCI_BYTEARRAY           Message,
                           PCI_BYTEARRAY           Data)
        struct strbuf ctlbuf;          /* stream message control part pointer*/
        struct strbuf databuf;                 /* stream message data part pointer    */
```

ExID identifies the STREAMS device. PCIMPB is a pointer to the PCI Message Parameter Block. Message and Data are the part of the PCI message to be sent to the NAF driver. Either Message or Data might be optional. In this case they are specified as NULL. In order to be more efficient (see STREAMS putmsg hereafter), it is recommended that the PCIMPB should be stored contiguously before the Message, this allows to avoid a copy in memory.

This function shall:

- prepare the ctlbuf and databuf structures;
- issue the putmsg() call;
- retrieve the error return;
- return appropriate error code.

STREAMS putmsg():

```
/* The general idea is to pass in ctlbuf a buffer containing the PCIMPB followed by the content of
Message, and in databuf the content of Data */
if (Message && ((char *)Message != (char *)PCIMPB + sizeof(pci_mpb))) {
        /*There is a Message not NULL, and PCIMPB and Message are not contiguous in memory, Have to
build a buffer where PCIMPB is followed by the Message content */
        char *buffer; /* pointer to a buffer, large enough to receive PCIMPB and the Message content */
        ...
        /* Here a memory allocation process may take place */
        ...
        memcpy (buffer, PCIMPB, sizeof(pci_mpb));
        memcpy ((buffer + sizeof(pci_mpb), Message, PCIMPB->MessageActualUsedSize);
        ctlbuf->buf    = buffer;
        ctlbuf->len    = PCIMPB->MessageActualUsedSize + sizeof(pci_mpb);
}
else {
        /* either there is no Message, or the PCIMPB and the Message are contiguous in memory */
        ctlbuf->buf    = PCIMPB;
        ctlbuf->len    = Message ? PCIMPB->MessageActualUsedSize + sizeof(pci_mpb): sizeof(pci_mpb);
}

databuf->buf= Data;
databuf->len= Data ? PCIMB->DataActualUsedSize: 0;
flags = 0;
if (putmsg (ExID, &ctlbuf, &databuf, flags) != 0) {
        /* Error condition, errno will be set       */
        ....
}
else {
        /* Operation OK     */
        ....
}
```

7.3.1.6.6 PciGetMessage

Function body:

```
PCI_INTEGER PciGetMessage (PCI_EXID          ExID,
                           struct pci_mpb    *PCIMPB,
                           PCI_BYTEARRA  Y   *Message,
                           PCI_BYTEARRAY     *Data)
        struct strbuf  ctlbuf;        /* stream message control part pointer */
        struct strbuf  databuf;             /* stream message data part pointer */
```

ExID identifies the STREAMS device. PCIMPB is a pointer to the PCI Message Parameter Block. Message and Data are the part of the PCI message to be received from the NAF driver. Either Message or Data might be optional. In this case they are specified as NULL. In order to be more efficient (see STREAMS getmsg hereafter), it is recommended that the PCIMPB be stored contiguously before the Message, this allows to avoid a copy in memory.

This function shall:

- prepare the **ctlbuf** and **databuf** structures;
- issue the getmsg () call;
- retrieve the return values from the **ctlbuf** and **databuf** structures;
- return appropriate error code.

STREAMS getmsg():

```
/* The general idea is to pass in ctlbuf a buffer large enough for containing the PCIMPB followed by the content of Message, and in databuf the content of Data. The error code of the NAF is available in the errno variable. */

if (Message && ((char *)Message != (char *)PCIMPB + sizeof(pci_mpb))) {
        /* there is a Message not NULL and, PCIMPB and Message are not contiguous in memory, have to reserve a buffer where PCIMPB can be followed by the Message content */
        char *buffer; /* pointer to a buffer,large enough to receive PCIMPB and the Message content */
        /* Here a memory allocation process may take place */
        ctlbuf->buf    = buffer;
}
else {
        /* either there is no Message, or the PCIMPB and the Message are contiguous in memory */
        ctlbuf->buf       =  PCIMPB;
}
ctlbuf->maxlen       = Message ? PCIMPB->MessageMaximumSize + sizeof(pci_mpb):sizeof(pci_mpb);
databuf->buf= Data;
databuf->maxlen    = Data ? PCIMPB->DataMaximumSize: 0;


flags = 0;
if (getmsg (ExID, &ctlbuf, &databuf, &flags) != 0) {
        /* Error condition, errno will be set        */
        PCIMPB->c_error = errno;
        ....
}
else { /* Operation OK     */
        if (ctlbuf->len != -1 && ctlbuf->len >= sizeof(pci_mpb)) {
                /* Message, possibly of size 0 is present */
                PCIMPB->MessageActualUsedSize     = ctlbuf->len - sizeof(pci_mpb);
                if (Message && ((char *)Message != (char *)PCIMPB + sizeof(pci_mpb)))
                        {
                        /* there is a Message not NULL and, PCIMPB and Message are not contiguous in memory, a buffer where PCIMPB is followed by the Message content, has been used */
                        memcpy (PCIMPB, buffer, sizeof(pci_mpb));
                        memcpy (Message,(buffer + sizeof(pci_mpb)), (ctlbuf->len - sizeof(pci_mpb)));
                }
                else {
                        /* the PCIMPB and the Message are contiguous in memory, no additional buffer used */
                        Message = PCIMPB + sizeof(pci_mpb);
                }
        }
        else {
        /* No Message present or too small message: error at least PCIMPB should be there */
        .............
        }


if (databuf->len != -1) {
        /* Data block, possibly of size 0 is present */
        PCIMPB->DataActualUsedSize = databuf->len;
}
else {
        /* No Data present */
        PCIMPB->DataActualUsedSize = 0;
}
}
```

7.3.1.6.7 PciSetSignal

Function body:

```
PCI_INTEGER     PciSetSignal(       PCI_EXID    *ExID,
                        PCI_INTEGER             Signal,
                        PCI_PROCEDURE           SignalProcedure)
{
        extern int              errno;
}
```

ExID identifies the STREAMS device, Signal the UNIX signal number. SignalProcedure is the address of the signal handler ('C' function) inside of the PUF. Only the UNIX SIGPOLL signal shall be issued by the NAF implementation. Consequently, any non-zero value in Signal shall turn on emission of UNIX SIGPOLL signals, a zero value shall turn emission off.

The SignalProcedure defined by the PUF shall reissue the signal via the signal() system call - see below. This mechanism is mandatory otherwise the next signal provided by the NAF shall kill the PUF.

More than one signal can be sent by a NAF to a PUF before the PUF accesses the NAF. An access to the NAF by the PUF during signal procedure treatment is not recommended.

This function shall:

- issue the ioctl() call;
- retrieve the error code;
- register UNIX SIGPOLL signalling with the stream head using: ioctl (..., I_SETSIG, S_MSG) system call;
- register UNIX SIGPOLL signalling with the operating system using: signal (SIGPOLL, SignalProcedure) system call;
- return appropriate error code.

STREAMS ioctl():

The function shall check the Signal parameter and shall, if Signal equals zero, set up the Signal_options variable to zero to turn off signalling. Furthermore, the signal function shall be de-registered by issuing the appropriate signal() call.

If Signal is non-zero, Signal_options shall be set to enable SIGPOLL signalling and any other options mandated by the implementation (see sigaction()). Furthermore, the signal function shall be registered using the signal() system call.

```
if (Signal == 0) {
        Signal_options = 0;
        if (ioctl (ExID, I_SETSIG, Signal_options) == -1)
                return errno;
        signal (SIGPOLL, SIG_DFL);
        return 0;
}
else {
        Signal_options = <SETSIG options>
        if (ioctl (ExID, I_SETSIG, Signal_options) == -1)
                return errno;
        signal (SIGPOLL, SignalProcedure);
        return 0;
}
```

7.3.1.7 Availability of NAF's PCI_HANDLE

To be accessible via the PciGetHandles function call, a NAF shall issue a declaration action. The inverse action - extraction from the list of available NAFs - is described too. These actions are operating system specific.

7.3.1.7.1 Declaration action

During the installation script of the STREAM driver, the directory **/etc/pcidd** is updated by a dummy file which is the name of the new NAF. The installation script may check availability of the NAF before the creation of the new dummy file.

7.3.1.7.2 Extraction action

During the de6sinstallation script of the STREAM driver, the directory **/etc/pcidd** is updated by removing the dummy file name of the NAF.

7.3.2 UNIX for Profile B

Profile B is incorporated in the UNIX environment as a kernel driver using streams facilities. Communication between such kernel drivers and applications are typically based on system calls **open**, **ioctl**, **putmsg**, **getmsg**, and **close**. To register at a device driver, an application opens a stream (*open()*), to deregister the system call *close()* is used. Data transfer from and to the driver is achieved by the calls *putmsg()* and *getmsg()*. Additional information exchange is done with the *ioctl()* system call.

Profile B uses this standardized driver access. Therefore the following specification does not define a complete functional interface (which will not be accepted by UNIX applications, which always are - and shall be - file I/O oriented). Instead Profile B's system call level interface will be introduced, which every UNIX like application can use to exchange Profile B messages and associated data. Of course, a functional interface can be offered (e.g. according to subclause 7.2), but that would not be the appropriate solution for an application interface for communication applications running under UNIX. Nevertheless, the following specification will offer the complete functionality of Profile B access operations used in other operating systems.

Profile B's device name is **/dev/capi20**. To allow multiple access of different UNIX processes, the device is realised as a clone streams device.

An application (in terms of Profile B) can register at Profile B (CAPI_REGISTER) by opening the device /dev/capi20 and issuing the relevant parameters via the system call *ioctl()* to the opened device. Note that the result of this operation is a file handle, not an application ID. So in the UNIX environment the application ID included in Profile B messages shall not be used to identify CAPI applications. The only valid handle between the Profile B kernel driver and the application based on a system call level interface is a UNIX file handle. To release from Profile B (CAPI_RELEASE), an application just closes the opened device. Profile B operations CAPI_PUT_MESSAGE and CAPI_GET_MESSAGE are achieved by system calls *putmsg()* and *getmsg()*. The functionality of CAPI_SET_SIGNAL need not be offered by Profile B; instead the UNIX signalling and/or waiting mechanism based on file descriptors can be used by applications. This includes the multiple wait on different file descriptors (*poll()*); a functionality which is not offered by Profile B based on other operating systems. Every other Profile B operation is realised by the system call *ioctl()* with appropriate parameters.

All messages are passed transparently through the UNIX driver interface.

To define the system call level interface in the UNIX environment, following data types imply following size:

ushort 16 bit unsigned integer;

unsigned 32 bit unsigned integer.

7.3.2.1 Message operations

7.3.2.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the three parameters maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs for the connections it is going to accept or it will try to establish itself.

CAPI_REGISTER	ioctl(): 0x01

Implementation
The following code fragment depicts the UNIX implementation of Profile B register functionality:

```
#include <sys/fcntl.h>                  /* open() parameters */
#include <sys/stropts.h>                /* streams ioctl() constants */
#include <sys/socket.h>                 /* streams ioctl() macros */
...
struct capi_register_params {
        unsigned    level3cnt;
        unsigned    datablkcnt;
        unsigned    datablklen;
} rp;
int fd;
struct strioctl strioctl;

/* open device */
fd = open("/dev/capi20", O_RDWR, 0);

 /* set register parameters */
rp.level3cnt = No. of simultaneous user data connections
rp.datablkcnt = No. of buffered data messages
rp.datablklen = Size of buffered data messages

 /* perform CAPI_REGISTER */
strioctl.ic_cmd = ('C' << 8) | 0x01;        /* CAPI_REGISTER */
strioctl.ic_timout = 0;
strioctl.ic_dp = (void *)(&rp);
strioctl.ic_len = sizeof(struct capi_register_params);
ioctl(fd, I_STR, &strioctl);
```

For simplicity, no error checking is shown in the example.

7.3.2.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. This way the application signals Profile B that all resources that have been allocated by Profile B for the application can be released again.

The application is identified by the application identification number that had been assigned in the previous CAPI_REGISTER operation.

CAPI_RELEASE	close()

Implementation
To release a connection between an application and Profile B driver, the system call *close()* is used. All related resources are released.

7.3.2.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an application identification number.

CAPI_PUT_MESSAGE	**putmsg()**

Implementation
To transfer a message from an application to Profile B driver and the controller behind, the system call *putmsg()* is used.

The application puts Profile B message into the ctl part of the *putmsg()* call. Parameter *data* and *data length* of message **DATA_B3_REQ** shall be stored in the data part of *putmsg()*.

NOTE: Profile B message is stored in the ctl part of *putmsg()*. In case of **DATA_B3_REQ** parameters *data* and *data length* in this ctl part of *putmsg()* are not interpreted from Profile B implementations.

7.3.2.1.4 CAPI_GET_MESSAGE

Description
With this operation the application retrieves a message from Profile B. The application retrieves all messages associated with the corresponding file descriptor from operation **CAPI_REGISTER**.

CAPI_GET_MESSAGE	**getmsg()**

Implementation
To receive a message from Profile B the application uses the system call *getmsg()*.

The application shall supply sufficient buffers for receiving the ctl and data parts of the message. In case of receiving Profile B message **DATA_B3_IND**, parameter *data* and *data length* of this message are not supported. Instead the data part of *getmsg()* is used to offer the transferred data.

NOTE: To receive a message from Profile B the application uses the system call *getmsg()*.

7.3.2.2 Other functions

7.3.2.2.1 CAPI_GET_MESSAGE

Description
With this operation the application determines the manufacturer identification of Profile B. The offered buffer shall have a size of at least 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

CAPI_GET_MANUFACTURER	ioctl(): 0x06

Implementation
This operation is realised using ioctl(0x06). The caller shall supply a buffer in struct strioctl ic_dp and ic_len.

```
int fd;                                 /* a valid Profile B handle */
struct strioctl strioctl;
char buffer[64];

strioctl.ic_cmd = ('C' << 8) | 0x06;    /* CAPI_GET_MANUFACTURER */
strioctl.ic_timout = 0;
strioctl.ic_dp = buffer;
strioctl.ic_len = sizeof(buffer);
ioctl(fd, I_STR, &strioctl);
```

The manufacturer identification is transferred to the given buffer. The string shall always be zero-terminated.

7.3.2.2.2 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number. The offered buffer shall have a size of 4 * sizeof(unsigned).

CAPI_GET_VERSION	ioctl(): 0x07

Implementation
This operation is realised using ioctl(0x07). The caller shall supply a buffer in struct strioctl ic_dp and ic_len.

```
int fd;                                 /* a valid Profile B handle */
struct strioctl strioctl;
unsigned unsigned buffer[4];

strioctl.ic_cmd = ('C' << 8) | 0x07;    /* CAPI_GET_VERSION */
strioctl.ic_timout = 0;
strioctl.ic_dp = buffer;
strioctl.ic_len = sizeof(buffer);
ioctl(fd, I_STR, &strioctl);
```

The buffer consists of four elements:

first	Profile B major version: 0x02
second	Profile B minor version: 0x00
third	manufacturer-specific major number
fourth	manufacturer-specific minor number

7.3.2.2.3 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. The offered buffer shall have a size of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

CAPI_GET_SERIAL_NUMBER	**ioctl(): 0x08**

Implementation
This operation is realised using ioctl(0x08). The caller shall supply a buffer in struct strioctl ic_dp and ic_len.

```
int fd;                                 /* a valid Profile B handle */
struct strioctl strioctl;
char buffer[8];

strioctl.ic_cmd = ('C' << 8) | 0x08;    /* CAPI_GET_SERIAL_NUMBER */
strioctl.ic_timout = 0;
strioctl.ic_dp = buffer;
strioctl.ic_len = sizeof(buffer);
ioctl(fd, I_STR, &strioctl);
```

The serial number consists of up to seven decimal-digit ASCII characters. It shall always be zero-terminated.

7.3.2.2.4 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. In the allocated buffer of 64 byte Profile B copies information about implemented features, number of controllers and supported protocols. *CtrlNr*, which is an input parameter for Profile B, is coded in the first bytes of the buffer and contains the controller number (bit 0..6), for which this information is requested.

CAPI_GET_PROFILE	**0x09**

Implementation
This operation is realised using ioctl(0x09). The caller shall supply a buffer in struct strioctl ic_dp and ic_len.

```
int fd;                                 /* a valid Profile B handle */
struct strioctl strioctl;
char buffer[64];

                                        /* Set Controller number */
* ((unsigned*)(&buffer[0])) = CtrlNr;

strioctl.ic_cmd = ('C' << 8) | 0x09;    /* CAPI_GET_PROFILE */
strioctl.ic_timout = 0;
strioctl.ic_dp = buffer;
strioctl.ic_len = sizeof(buffer);
ioctl(fd, I_STR, &strioctl);
```

Structure of command specific parameters:

Parameter	Comment
CtrlNr	Number of Controller. If 0, only number of installed controllers is given to the application.

Retrieved structure format:

Type	Description
ushort	number of installed controllers, least significant octet first
ushort	number of supported B-channels, least significant octet first
unsigned	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4.[31]: reserved
unsigned	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
unsigned	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
unsigned	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
6 unsigned	reserved for Profile B usage
5 unsigned	manufacturer specific information
NOTE: This function can be extended, so an application shall ignore unknown bits. Profile B shall set every reserved field to 0.	

7.4 OS/2

7.4.1 OS/2 Operation System specific implementation for Profile A

7.4.1.1 Introduction

This subclause describes the operating system specific implementation for the DOS operating system. For the following description, the OS/2™ version starts with the version 2.0.

A NAF implementation under 0S/2 shall offer the functionality of the exchange functions described in a generic way in subclause 5.3.

In this subclause, the mapping and implementation of these functions is described on a function per function basis using the "C" language.

7.4.1.2 OS/2 application level

For operating system OS/2 Version 2.x an application program can access Profile A services via a DLL (Dynamic Link Library). In this case the NAF shall be a DLL.

7.4.1.2.1 Mechanism

Except for the PciGetHandles function call, the DLL mechanism is the basic mechanism used to support the Profile A exchange method under OS/2. At OS/2 application level every NAF shall be a 32 bit DLL and shall export an entry point per Profile A function using the same name: PciGetProperty (ordinal number = 1), PciRegister (ordinal number = 2), PciDeregister (ordinal number = 3), PciGetMessage (ordinal number = 4), PciPutMessage (ordinal number = 5), PciSetSignal (ordinal number = 6).

NOTE: Function name exported by the NAF are the same than the description made in subclause 5.3 but parameters are different.

PciGetHandles needs an access to the PCI.INI file.

PciRegister and PciGetProperty check if the DLL, accessible by its name, is available.

To access a NAF the only need for a PUF is to know the name of the DLL. The address access to the DLL may be provided transparently to the PUF inside the Pci's exchange mechanism functions as described in the annex J.

The PciRegister function dynamically loads the NAF. It needs to keep trace of the handle of the NAF as a DLL, so this handle is part of the Exchange Identifier. The NAF need also to keep trace of the PUF, so it assigns an Identifier to the PUF at registration time, this NAF-provided Identifier is the other part of the Exchange Identifier.

The common calling conventions to provide parameter to a DLL is the PASCAL calling convention. This convention is also used by the Profile A exchange method under OS/2.

Pointer parameters are far. The PCIMPB structure is always passed via a pointer. The Exchange Identifier structure is always passed via a pointer.

The structure alignment is **byte**.

7.4.1.2.2 Implementation of basic type

Under OS/2, the following values shall be used:

PCI_HANDLE	name of the DLL
PCI_EXID	Structure contents handle provided by OS/2 the DLL is loaded Unique Identifier provided by NAF to identify the PUF
PCI_PROCEDURE	exported function address provided by the PUF
PCI_INTEGER	2 bytes
PCI_BYTEARRAY	far pointer

7.4.1.2.3 C Function prototypes

```
/*
 * Basic types
 */
typedef SHORT                   PCI_INTEGER;
typedef PSZ                     PCI_BYTEARRAY;
typedef PSZ                     PCI_HANDLE;
typedef struct                  {
                                HMODULE         hDLLInstance;
                                PCI_INTEGER     Exchange_Id;
                                } PCI_EXID;
typedef PSZ                     PCI_PROCEDURE;


/*
 * Structures
 */
struct pci_mpb {
            PCI_INTEGER     MessageID;
            PCI_INTEGER     MessageMaximumSize;
            PCI_INTEGER     MessageActualUsedSize;
            PCI_INTEGER     DataMaximumSize;
            PCI_INTEGER     DataActualUsedSize;
};

struct pci_register {                   /* structure containing registering info */
        PCI_INTEGER PUFVersion;     /* optional: give PUF version */
        PCI_INTEGER PUFType;        /* optional: give PUF type */
        PCI_INTEGER MaxMsgSize;     /* return: max size of a message */
};

struct pci_opsys {          /* structure containing specific operating system info */
        int     DummyParameter;/* No specific requirement for OS2 */
};

/*
 * Exchange functions prototypes
 */
PCI_INTEGER far PASCAL PciGetHandles (      PCI_INTEGER MaxHandles,
                            PCI_BYTEARRAY PCIHandles,
                            PCI_INTEGER far * ActualHandles);

PCI_INTEGER far PASCAL PciGetProperty (     PCI_HANDLE PCIHandle,
                            PCI_INTEGER MaximumSize,
                            PCI_BYTEARRAY Property,
                            PCI_INTEGER far * ActualSize);
```

```
PCI_INTEGER far PASCAL PciRegister (     PCI_HANDLE PCIHandle,
                        struct pci_register * PCIRegisterInfo,
                        struct pci_opsys * PCIOpSysInfo,
                        PCI_EXID far *ExID);

PCI_INTEGER far PASCAL PciDeregister (  PCI_EXID far *ExID);

PCI_INTEGER far PASCAL PciPutMessage (       PCI_EXID far *ExID,
                        struct pci_mpb far *PCIMPB,
                        PCI_BYTEARRAY Message,
                        PCI_BYTEARRAY Data);

PCI_INTEGER far PASCAL PciGetMessage (       PCI_EXID far *ExID,
                        struct pci_mpb far *PCIMPB,
                        PCI_BYTEARRAY Message,
                        PCI_BYTEARRAY Data);

PCI_INTEGER far PASCAL PciSetSignal (    PCI_EXID far *ExID,
                        PCI_INTEGER Signal,
                        PCI_PROCEDURE SignalProcedure);
```

7.4.1.2.4 Description of functions

This subclause describes the implementation, under OS/2, of the Profile A exchange method functions. During a PUF to NAF call, the size of the stack shall be at least 1 024 bytes deep.

7.4.1.2.4.1 PciGetHandles

Under OS/2, the PciGetHandles uses a PCI.INI file in the OS/2 directory to get available PCI_HANDLEs.

The section [PCI_DLL] in the PCI.INI file contains all entries of installed NAFs. Each entry has the format:

pciDLL<number>=DLLName (number=1..32)

The following operations shall get all names of installed NAF drivers:

- loops from 1 to 32
  - constructs of the keyName 'pciDLL' associated to the current loop value;
  - issue a PrfQueryProfileString using:

  sectionKey = 'PCI_DLL',
  the keyName constructs before,
  no default value,
  a maximum size equal to 254,
  FileName = 'PCI.INI'.

7.4.1.2.4.2 PciGetProperty

This function is in charge to provide to the PUF the PROPERTY of the NAF. Implicitly it checks if the NAF is available - loading the library via the DosLoadModule function.

The following operations shall take place, in order:

- load the DLL;
- get the address of the PciGetProperty function exported by the NAF (DosQueryProcAddr);
- call to this address with the parameters provided by the PUF;
- free the loaded library.

7.4.1.2.4.3 PciRegister

This function is in charge to provide an association between a PUF and a NAF. The NAF is loaded and the Exchange Identifier is provided. The availability of the chosen NAF is checked during the load of the library. The library is identified by its name. Parameters for the registration operation are brought together in a structure:

- PUFType (PCI_INTEGER);
- PUFVersion (PCI_INTERGER);
- MaxMsgSize (PCI_INTEGER) where the NAF will give the maximum size for a message.

The following operations shall take place, in order:

- load the DLL;
- provide the DLLInstance part of the Exchange Identifier with the DLL Instance;
- get the address of the PciRegister function exported by the NAF (DosQueryProcAddr);
- call to this address to inform the NAF of a new PUF. The address of the registration parameters structure and the address of the Exchange Identifier structure are passed to the NAF as parameters;
- on return from the NAF, the Exchange_Id part of the Exchange Identifier and the maximum message size parameter of the registration parameter structure have been provided by the NAF;
- return to the PUF with the return code from the NAF.

7.4.1.2.4.4 PciDeregister

This function is in charge of disassociating a PUF and a NAF. The DLL is not freed from the memory each time a PUF deregisters a NAF but only when the last PUF deregisters the NAF.

The following operations shall take place, in order:

a) get the address of the PciDeregister function exported by the NAF (DosQueryProcAddr);
b) call to this address to inform the NAF of the end of the association. The PCI_EXID is passed to the NAF by address;
c) free the DLL.

7.4.1.2.4.5 PciPutMessage

This function is in charge to provide a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as in the description of the PciPutMessage.

The following operations shall take place, in order:

- get the address of the PciPutMessage function exported by the NAF (DosQueryProcAddr);
- call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.4.1.2.4.6 PciGetMessage

This function is in charge to provide a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as in the description of the PciGetMessage. Buffers provided by the PUF are directly used by the NAF.

The following operations shall take place, in order:

- get the address of the PciGetMessage function exported by the NAF (DosQueryProcAddr);
- call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.4.1.2.4.7 PciSetSignal

This function allows a PUF to provide a direct information mechanism to be used by the NAF in case of incoming event.

Under OS/2, the mechanism is based on a 32 bit system semaphore.

The following operations shall take place, in order:

- create a system semaphore by calling the DosCreateEventSem() function; this function provides a semaphore handle on return;

- get the address of the PciSetSignal function exported by the NAF (DosQueryProcAddr);

- call this address to pass parameters to the NAF (including the address of the PCI_EXID); the SignalProcedure parameter contains the system semaphore handle; the Signal parameter is not used but the parameter shall be passed to the NAF with the 0 value.

To signal an incoming event, the NAF post the system semaphore (DosPostEventSem) increasing a post-count value that is associated to the semaphore.

The PUF application thread may wait (DosWaitEventSem) until the post-count of the system semaphore is larger than 0. The PUF application can determine the current post count and simultaneously reset the post count by calling the DosResetEventSem() function.

To deactivate the mechanism the PciSetSignal function Signal and SignalProcedure parameters shall be NULL. Once deactivated, the previous mechanism shall no longer be used by the NAF to call the PUF.

On return from the NAF, the PUF application thread closes the system semaphore (DosCloseEventSem).

7.4.1.3 OS/2 device driver level

For operating system OS/2 Version 2.x application in form of OS/2 physical device driver (PDD) can access Profile A services via the Inter Device Driver Interface. In this case the NAF have to be a physical character or block device driver.

7.4.1.3.1 Mechanism

Except for the function **PciGetHandles**, the implementation of the exchange method for OS/2 physical device driver is based on a direct access mechanism. The access point is a far function address provided by the NAF. This function address is mapped to the generic type PCI_HANDLE.

A NAF offers its services to a PUF in form of an OS/2 physical device driver via the Inter Device Driver Interface. An application PDD issues an Inter Device Driver call (IDC) to execute operations.

Only one access point shall be provided by the NAF. A supplied parameter shall indicate the function to be invoked. This parameter is named **function code**.

Parameters are passed from the PUF to the NAF using the stack. The PUF shall ensure a minimum stack space of 128 bytes on call. When the NAF receives the control of the CPU, the first parameter on the stack is the function code, followed by parameters based on the particular function.

The function code is passed as a 2 byte integer value.

The NAF has to place the return code in the AX register. The NAF procedure is not in charge of cleaning the stack on return. The C call convention is used: the calling PUF pushes parameters right to left and restores the stack on return.

The alignment of the PCIMPB generic structure is **byte**.

Under OS/2 2.x every Physical Device Driver are 16:16 segment modules, thus all functions described in this clause are 16 bit functions and all pointers are 16:16 segmented.

The NAF PDD name shall be eight characters in length (blank extended to 8 characters) and is contained in its device driver header. The NAF PDD header shall contain the offset to its Inter Device Driver call entry point. The IDC bit of the Device Attribute Field in the device header shall be set to 1. The passed memory in parameters shall be either fixed or locked.

7.4.1.3.2 Implementation of basic types

Under OS/2, the following values shall be used:

PCI_HANDLE	name of the device driver (segment: offset address)
PCI_EXID	Structure contents handle provided by OS/2 when the DLL is loaded Unique Identifier provided by NAF to identify the PUF DS value of the NAF provided by the AttachDD device help call
PCI_PROCEDURE	exported function address provided by the PUF
PCI_INTEGER	2 bytes
PCI_BYTEARRAY	far pointer

All values are in little endian (low byte - high byte) order.

The **function code**, used to invoke the exchange functions, shall be assigned as follows:

Function	Function code value
PciGetProperty	1
PciRegister	2
PciDeregister	3
PciPutMessage	4
PciGetMessage	5
PciSetSignal	6

A C-language presentation of these definitions looks as follows:

```
/*
 * Generic type mappings
 */
typedef short int           PCI_INTEGER;
typedef char far *          PCI_BYTEARRAY;
typedef struct {
        short int           (far *hNaf)();
        PCI_INTEGER         Exchange_Id;
        unsigned short      Naf_Ds;

        } PCI_EXID;
typedef short int           PCI_EXID;
typedef char far *          PCI_HANDLE;
typedef void                (far *  PCI_PROCEDURE) ();

/*
 * Function code constants
 */
#define     PCIGETPROPERTY          1
#define     PCIREGISTER             2
#define     PCIDEREGISTER           3
#define     PCIPUTMESSAGE           4
#define     PCIGETMESSAGE           5
#define     PCISETSIGNAL            6
```

7.4.1.3.3 Description of functions

The PUF is in charge to provide a minimal stack during a function call. The minimal stack size is 128 bytes.

7.4.1.3.3.1 PciGetHandles

Under OS/2, the implementation of the **PciGetHandles** function shall use a PCI.INI file in the OS/2 directory to get available PCI_HANDLEs. This function call is the exception on the basic principle - direct access - under OS/2.

This implementation is the same as the application level implementation of the PciGetHandles function described in subclause 7.4.1.2.4.1.

The section [PCI_PDD] in the PCI.INI file contains all entries of installed NAFs. Each entry has the format:

pciDriver<number>=DriverName (number=1..32)

The following operations shall get all names of installed NAF drivers:

- loops from 1 to 32
  - constructs of the keyName 'pciDriver' associated to the current loop value;
  - issue a PrfQueryProfileString using:
    sectionKey = 'PCI_PDD',
    the keyName constructs before,
    no default value,
    a maximum size equal to 8,
    FileName = 'PCI.INI'.

7.4.1.3.3.2 PciGetProperty

This function is in charge of retrieving the NAF-Property from the NAF. To issue the function call, the PUF shall have knowledge of the PCI-Handle of the NAF it wants to access to.

The following operation shall be carried out by the PUF, in order:

- gain access to the NAF PDD by issuing an AttachDD device help call. This call returns the protected mode IDC entry point as a 16:16 segmented pointer and the data segment of the NAF PDD;

- call the address with the PciGetProperty **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately;

- check return code.

7.4.1.3.3.3 PciRegister

This function is in charge of providing an association between a PUF and a NAF. To issue the function call, the PUF shall have the knowledge of the PCI-Handle of the NAF it wants to access Before accessing the NAF, the PUF shall check, if the PCI-Handle it uses is valid by checking the signature of the access point the PCI-Handle is pointing to.

For this function call, two structures shall be prepared by the PUF and shall be passed on the function stack. The first structure is the PCIRegisterInfo structure as declared in subclause 5.3. The second is the operating system dependent PCIOpSysInfo structure, which for OS/2 has the following layout:

Structure Element Name	Type	Valid on Call or Return	Explanation
MaxNCOCount	2 byte integer	call	Shall be set to the maximum amount of NCOs the PUF intends to create during the association.
MaxPacketSize	2 byte integer	call	Shall be set to the maximum size of a data packet the NAF shall accept on a user connection.
MaxPacketCount	2 byte integer	call	Shall be set to the maximum amount of packets of the above size the NAF shall buffer per user connection.
AddBufferSize	4 byte integer	call	If the PUF wants to provide buffer space to the NAF, it shall set this value to the size of the buffer space it donates. Otherwise the value shall be set to zero (0).
AddBufferSpace	far address (segment: offset)	call	If the structure element AddBufferSize is non-zero, this element shall point (far) to the donated, additional buffer space.
BufferNeeded	4 byte integer	return	In case the NAF has not enough buffer space available to guarantee the requested connection characteristics, the amount of additional buffer needed is returned into this element by the NAF.

The information provided with this structure helps the NAF to optimise its internal resources. Therefore the information given by the PUF shall be carefully weighted. This is especially true in an environment, where a NAF is linked to several PUFs at the same time.

If a NAF does not have available enough memory resources to fulfil the requested characteristics, the PciRegister function will fail and return a BufferTooSmall error code. In this case the amount of buffer missing can be taken from the BufferNeeded element of the above structure.

On successful return of the PciRegister function, the Exchange-ID becomes available, which shall be used as a parameter on subsequent exchange mechanism function calls.

The following operation shall be carried out by the PUF, in order:

- gain access to the NAF PDD by issuing an AttachDD device help call. This call returns the protected mode IDC entry point as a 16:16 segmented pointer and the data segment of the NAF PDD;

- allocate and set-up the two structures PCIRegisterInfo and PCIOpSysInfo. The PCIOpSysInfo structure may optionally contain a pointer to additional buffer space which shall be donated to the NAF

- call the address with the PciGetProperty **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately;

- check return code. If the return code indicates OutOfBuffers then the call may be repeated with correct adjusted buffer space to be donated to the NAF;

- keep the returned Exchange-ID for later calling.

7.4.1.3.3.4 PciDeregister

This function is in charge to disassociate a PUF and a NAF.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciDeregister **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately;

- Check the return code.

7.4.1.3.3.5 PciPutMessage

This function is in charge of providing a message from a PUF to a NAF. Parameters shall be provided in the same order as indicated in the generic description of the PciPutMessage function.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciPutMessage **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately. The passed memory in parameters shall be either fixed or locked;

- Check the return code.

7.4.1.3.3.6 PciGetMessage

This function is in charge of providing the PUF with a message coming from the NAF. Parameters shall be provided in the same order as indicated in the generic description of the PciGetMessage function.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciGetMessage **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately;
- check the return code.

7.4.1.3.3.7 PciSetSignal

This function is in charge of providing the NAF with the address of a function located inside the PUF, which shall be called-back if a message becomes available for the PUF.

The following operation shall be carried out by the PUF, in order:

- call the address with the PciSetSignal **function code** and the parameters provided by the PUF. Before calling the IDC entry point of the NAF PDD, the application PUF PDD shall set-up the data segment register DS appropriately;

- check the return code

The NAF calls-back the PUF with the following conventions applying:

a) the NAF provides a minimal stack size of 128 bytes;
b) the values of the DS and ES segments are undefined;
c) interrupts are disabled.

Gaining control, the PUF:

1) may or may not enable interrupts;

2) is allowed call the NAF via the PciGetMessage or the PciPutMessage function;

3) shall not invoke other exchange function calls besides the PciGetMessage and the PciPutMessage functions;

4) shall not issue OS/2 system calls;

5) shall not let interrupts disabled over an extended period of time and shall return from the call-back function as quick as possible.

The NAF called via the PciGetMessage or the PciPutMessage function may enable interrupts. However, the NAF shall not call the call-back routine again, until the call-back routine has returned normally.

At the end of the call-back routine the PUF shall return to the NAF. Only the SS:SP register pair shall be preserved by the PUF.

To deactivate the mechanism the PciSetSignal function Signal and SignalProcedure parameters shall be NULL. Once deactivated, the previous mechanism shall no longer be used by the NAF to call the PUF.

7.4.1.4 NAF availability

7.4.1.4.1 Declaration action

First, the NAF may get the list of available PCI_HANDLEs to check if not already declared. The mechanism the NAF uses is the same as any PUF to get available NAF: PciGetHandles (see subclauses 7.4.1.2.4.1 and 7.4.1.3.3.1).

If not yet declared, the NAF includes its own PCI_HANDLE into the list.

```
PCI_BYTEARRAY ownDLLName = "xxx";
#ifdef PCI_DLL              /* For OS/2 application level */
PCI_BYTE          driverName[254];
#endif
#ifdef PCI_PDD              /* For OS/2 device driver level */
PCI_BYTE          driverName[8];
#endif
WORD                        index;
char                        keyName[20];
HINI                        hini;

hini = PrfOpenProfile(hini, "PCI.INI);

/* Check if NAF not already installed */
for (index = 1; index <=  32; index++)
        {
#ifdef PCI_DLL                              /* For OS/2 application level */
        sprintf(keyName, "pciDLL%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_DLL",  /* Section name */
#endif
#ifdef PCI_PDD                              /* For OS/2 device driver level */
        sprintf(keyName, "pciDriver%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_PDD", /* Section name */
#endif
                                    keyName,    /* "pciDriver"+1..n */
                                    NULL,               /* No default needed */
                                    driverName,
                                    sizeof(driverName)) > 0)
                {
                if (strcmpi(driverName, ownDLLName) == 0) return; /* NAFinstalled, OK return */
                }
        }

/* Search a free pciDriver position */
for (index = 1, index <= 32; index++)
        {
```

```
#ifdef PCI_DLL                          /* For OS/2 application level */
        sprintf(keyName, "pciDLL%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_DLL",  /* Section name */
#endif
#ifdef PCI_PDD                          /* For OS/2 device driver level */
        sprintf(keyName, "pciDriver%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_PDD", /* Section name */
#endif
                                    keyName,    /* "pciDriver"+1..n */
                                    NULL,               /* No default needed */
                                    driverName,
                                    sizeof(driverName),
                                    "PCI.INI") == 0)
                {
                /* Entry does not exist, add own NAF Driver name */
#ifdef PCI_DLL                          /* For OS/2 application level */
                PrfWriteProfileString(hini, "PCI_DLL", keyName, ownDLLName);
#endif
#ifdef PCI_PDD                          /* For OS/2 device driver level */
                PrfWriteProfileString(hini, "PCI_PDD", keyName, ownDLLName);
#endif
                return;
                }
        }

PrfCloseProfile(hini);
```

Maximum number of NAF than can register is 32.

7.4.1.4.2 Extraction action

First, the NAF gets the list of available PCI_HANDLEs to check if it is declared. If so, the NAF removes its own PCI_HANDLE from the driver list in "PCI.INI".

```
PCI_BYTEARRAY ownDLLName = "xxx";
#ifdef PCI_DLL          /* For OS/2 application level */
PCI_BYTE        driverName[254];
#endif
#ifdef PCI_PDD          /* For OS/2 device driver level */
PCI_BYTE        driverName[8];
#endif
WORD                    index;
char                    keyName[20];
HINI                    hini;

hini = PrfOpenProfile(hini, "PCI.INI);

for (index = 1, index <= 32; index++)
        {
#ifdef PCI_DLL                          /* For OS/2 application level */
        sprintf(keyName, "pciDLL%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_DLL",  /* Section name */
#endif
#ifdef PCI_PDD                          /* For OS/2 device driver level */
        sprintf(keyName, "pciDriver%d", index);
        if (PrfQueryProfileString(  hini,
                                    "PCI_PDD", /* Section name */
```

```
#endif
                                keyName,    /* "pciDriver"+1..n
                                NULL,               /* No default needed */
                                driverName,
                                sizeof(driverName),
                                "PCI.INI") > 0)
            {
            /* Check for own name */
            if (strcmpi(driverName, ownDLLName) == 0)
                {
                /* Remove the name of the Driver */
#ifdef PCI_DLL                          /* For OS/2 application level */
                PrfWriteProfileString(hini, "PCI_DLL", keyName, "");
#endif
#ifdef PCI_PDD                          /* For OS/2 device driver level */
                PrfWriteProfileString(hini, "PCI_PDD", keyName, "");
#endif
                }
            }
        }

    PrfCloseProfile(hini);
```

7.4.2 OS/2 for Profile B

7.4.2.1 OS/2 (application level)

In a PC environment with operating system OS/2 Version 2.x an application program can access Profile B services via a DLL (Dynamic Link Library). The interface between applications and Profile B is realised as a function interface. An application can issue Profile B function calls to perform Profile B operations.

The DLL providing the function interface shall be named "CAPI20.DLL". It is a 32 bit DLL exporting 32 bit functions with System-Call-Convention. This means all parameters are pushed on the stack, the calling process shall clear up the stack after it returns from the function call.

The functions are exported under following names and ordinal numbers:

CAPI_MANUFACTURER (reserved)	CAPI20.99
CAPI_REGISTER	CAPI20.1
CAPI_RELEASE	CAPI20.2
CAPI_PUT_MESSAGE	CAPI20.3
CAPI_GET_MESSAGE	CAPI20.4
CAPI_SET_SIGNAL	CAPI20.5
CAPI_GET_MANUFACTURER	CAPI20.6
CAPI_GET_VERSION	CAPI20.7
CAPI_GET_SERIAL_NUMBER	CAPI20.8
CAPI_GET_PROFILE	CAPI20.9
CAPI_INSTALLED	CAPI20.10

These functions can be called by an application according to the DLL conventions as imported functions. If an application calls the DLL it shall ensure that there are at least 512 bytes left on the stack.

All pointers that are passed from the application program to Profile B, or vice versa, in function calls or in messages are 0:32 flat pointers. This especially applies to the data pointer in **DATA_B3_REQ** and **DATA_B3_IND** messages. The referenced data shall not cross a 64 kByte boundary in the flat address space because the DLL may convert the passed flat pointer to a 16:16 bit segmented pointer.

In the OS/2 environment following types are used to define the functional interface:

word	16 bit unsigned integer
dword	32 bit unsigned integer
void*	0:32 flat pointer to any memory location
void**	0:32 flat pointer to a void *
char*	0:32 flat pointer to a character string
dword*	0:32 flat pointer to a 32 bit unsigned integer value

7.4.2.1.1 Message operations

7.4.2.1.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the four parameters messageBufferSize, maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs.

For a 'normal' application the size of the message buffer should be calculated using the following formula:

MessageBufferSize = 1024 + (1024 * maxLogicalConnection)

Function call

```
dword FAR PASCAL CAPI_REGISTER (    dword messageBufferSize,
                                    dword maxLogicalConnection,
                                    dword maxBDataBlocks,
                                    dword maxBDataLen,
                                    dword* pApplID);
```

Parameter	Comment
messageBufferSize	Size of Message Buffer
maxLogicalConnection	Maximum number of logical connections
maxBDataBlocks	Number of data blocks available simultaneously
maxBDataLen	Maximum size of a data block
pApplID	Pointer to the location where Profile B should place the assigned application identification number

Return Value

Return Value	Comment
0x0000	Registration successful - application identification number has been assigned
All other values	Coded as described in parameter info class 0x10xx

7.4.2.1.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. Profile B shall release all resources that have been allocated.

The application is identified by the application identification number that had been assigned in the previous CAPI_REGISTER operation.

Function call

dword FAR PASCAL CAPI_RELEASE (dword ApplID);

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER

Return Value

Return Value	Comment
0x0000	Release of the application successful
All other values	Coded as described in parameter info class 0x11xx

7.4.2.1.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an application identification number. The message memory area shall not cross a 64 kByte boundary (e.g. use *tiled* memory) in the flat address space because the DLL may convert the passed flat pointer to a 16:16 bit segmented pointer. The same applies to B3 data blocks that are passed within DATA_B3_REQ messages.

Function call

dword FAR PASCAL CAPI_PUT_MESSAGE (dword ApplID, **void* pCAPIMessage);**

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
pCAPIMessage	0:32 (flat) pointer to the message that is passed to Profile B

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx
NOTE: When the process returns from the function call the message memory area can be reused by the application.	

7.4.2.1.1.4 CAPI_GET_MESSAGE

Description
With this operation the application retrieves a message from Profile B. The application can only retrieve those messages intended for the stipulated application identification number. If there is no message waiting for retrieval, the function returns immediately with an error code.

Function call

dword FAR PASCAL CAPI_GET_MESSAGE (	**dword ApplID,** **void** ppCAPIMessage);**

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
ppCAPIMessage	0:32 (flat) pointer to the memory location where Profile B should place the 0:32 (flat) pointer to the retrieved message

Return Value

Return Value	Comment
0x0000	Successful - Message was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx
NOTE: The received message may become invalid the next time the application issues a **CAPI_GET_MESSAGE** operation for the same application identification number. This especially matters in multi threaded applications where more than one thread may execute **CAPI_GET_MESSAGE** operations. The synchronisation between threads shall be done by the application.	

7.4.2.1.2 Other functions

7.4.2.1.2.1 CAPI_SET_SIGNAL

Description
This operation is used by the application to install a mechanism which signals the application the availability of a message.

In OS/2 2.x this is done best by using a fast 32 bit system event semaphore. The application shall create the used semaphore by calling the *DosCreateEventSem()* function which is part of the OS/2 system application program interface. This routine provides a semaphore handle which is passed as a parameter in the CAPI_SET_SIGNAL call.

In that case each time Profile B places a message in the application's message queue the specified semaphore is "posted" increasing a post-count value that is associated to the semaphore. To do so Profile B executes the *DosPostEventSem()* function of the OS/2 system API.

The application thread may wait until the post-count of the semaphore is larger than 0 using the *DosWaitEventSem()* OS/2 system call. It can determine the current post count and simultaneously reset the post count executing the *DosResetEventSem()* OS/2 system API call.

By issuing this function call with a semaphore handle of 0 the signalling mechanism is deactivated.

Function call

```
dword FAR PASCAL CAPI_SET_SIGNAL (      dword ApplID,
                                        dword hEventSem);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
hEventSem	Event Semaphore handle assigned by operating system

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

7.4.2.1.2.2 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B (DLL). SzBuffer on call is a 0:32 (flat) pointer to a buffer of 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

Function call

```
void FAR PASCAL CAPI_GET_MANUFACTURER (char* SzBuffer);
```

Parameter	Comment
SzBuffer	0:32 (flat) pointer to a buffer of 64 bytes

7.4.2.1.2.3 CAPI_GET_MANUFACTURER

Description
With this function the application determines the version of Profile B as well as an internal revision number.

Function call

```
dword FAR PASCAL CAPI_GET_VERSION (     dword* pCAPIMajor,
                                        dword* pCAPIMinor,
                                        dword* pManufacturerMajor,
                                        dword* pManufacturerMinor);
```

Parameter	Comment
pCAPIMajor	0:32 (flat) protected mode pointer to a dword receiving Profile B major version number: 2
pCAPIMinor	0:32 (flat) protected mode pointer to a dword receiving Profile B minor version number: 0
pManufacturerMajor	0:32 (flat) protected mode pointer to a dword receiving manufacturer specific major number
pManufacturerMinor	0:32 (flat) protected mode pointer to a dword receiving manufacturer specific minor number

Return Value

Return	Comment
0x0000	No error, version numbers are copied.

7.4.2.1.2.4 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. SzBuffer on call is a 0:32 (segmented) protected mode pointer to a buffer of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

Function call

```
dword FAR PASCAL CAPI_GET_SERIAL_NUMBER (char* SzBuffer);
```

Parameter	Comment
SzBuffer	0:32 (flat) pointer to a buffer of 8 bytes

Return Value

Return	Comment
0x0000	No error SzBuffer contains the serial number in plain text in the form of a 7-digit number. If no serial number is provided by the manufacturer, an empty string is returned.

7.4.2.1.2.5 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. SzBuffer on call is a 0:32 (flat) protected mode pointer to a buffer of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. *CtrlNr* contains the controller number (bit 0..6), for which this information is requested.

```
dword FAR PASCAL CAPI_GET_PROFILE (     LPBYTE SzBuffer,
                                        WORD CtrlNr
                                        );
```

Parameter	Comment
SzBuffer	0:32 (flat) protected mode pointer to a buffer of 64 bytes
CtrlNr	Number of Controller. If 0, only number of installed controllers is given to the application.

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
WORD	number of installed controllers, least significant octet first
WORD	number of supported B-channels, least significant octet first
DWORD	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4.[31]: reserved
DWORD	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
DWORD	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
DWORD	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for fax group 3 5..31: reserved
6 DWORDs	reserved for Profile B usage
5 DWORDs	manufacturer specific information
NOTE: This function can be extended, so an application has to ignore unknown bits. Profile B shall set every reserved field to 0.	

7.4.2.1.2.6 CAPI_INSTALLED

Description
This function can be ued by an application to determine if the ISDN hardware and necessary drivers are installed.

Function call

dword FAR PASCAL CAPI_INSTALLED (void)

Return Value

Return	Comment
0x0000	Profile B is installed
Any other value	Coded as described in parameter info class 0x11xx

7.4.2.2 OS/2 (device driver level)

In a PC environment with operating system OS/2 Version 2.x there may exist Profile B applications in form of OS/2 Physical Device Drivers (PDD). Those applications are referred as "application PDDs" in the following sections. This specification describes the interface of an OS/2 2.x physical device driver offering Profile B services to other device drivers. Profile B PDD is called "CAPI PDD" in the following sections.

PDDs under OS/2 2.x are 16:16 segment modules, thus all functions in this specification are 16 bit functions, all pointers are 16:16 segmented.

In this subclause following data types are used to define the interface:

word	16 bit unsigned integer
dword	32 bit unsigned integer
void*	16:16 (segmented) pointer to any memory location
void**	16:16 (segmented) pointer to a void*
char*	16:16 (segmented) pointer to a character string
word*	16:16 (segmented) pointer to a word

The CAPI PDD offers its services to application PDDs via the Inter Device Driver Interface. An application PDD issues an Inter Device Driver Call (IDC) to execute CAPI operations.

The CAPI PDD name which is contained in its device driver header shall be "CAPI20 " (blank extended to 8 characters). The CAPI PDD header shall contain the offset to its inter device driver call entry point. The IDC bit of the Device Attribute Field in the device driver header shall be set to 1.

An application PDD gains access to the CAPI PDD by issuing an *AttachDD* device help call. This call returns the protected mode IDC entry point as a 16:16 segmented pointer and the data segment of the CAPI PDD. Before calling the IDC entry point of the CAPI PDD the application PDD shall set-up the data segment register DS appropriately.

This is the prototype of the CAPI PDD IDC function:

word CAPI20_IDC (word funcCode, void *funcPara);

The function is called with "C" calling convention, thus the calling application PDD shall clear up the stack. When the application PDD calls the IDC function there shall be at least a space of 512 bytes left on the stack. The parameter funcCode selects the CAPI operation to take place, the parameter funcPara contains a 16:16 segmented pointer to the CAPI operation specific parameters. The structure of these parameters is defined in the following subclauses. The function returns an error code which is 0 if no error occurred. Which CAPI operations may cause which error codes is also defined in the following subclauses.

7.4.2.2.1 Message operations

7.4.2.2.1.1 CAPI_REGISTER

Description
This is the operation the application PDD uses to report its presence to Profile B. By passing the four parameters messageBufferSize, maxLogicalConnection, maxBDataBlocks and maxBDataLen the application PDD describes its needs. By use of the parameter Buffer the application PDD passes a memory area to Profile B. Profile B uses this memory area to store messages and data blocks destined to the application PDD. The passed memory shall be either fixed or locked. Profile B does not need to verify if this storage really exists.

The size of the memory area shall be calculated according to the following formula:

MessageBufferSize + (maxLogicalConnection * maxBDataBlocks * maxBDataLen)

Choosing too small a value can result in messages being lost. The size of the message buffer should be calculated for a "normal" application PDD according to following formula:

MessageBufferSize = 1024 + (1024 * maxLogicalConnection)

CAPI_REGISTER	**0x01**

Structure of command specific parameters:

Parameter	Type	Comment
Buffer	void*	16:16 (segmented) pointer to a memory region provided by the application PDD. Profile B uses this memory area to store messages and data blocks destined for the application PDD.
messageBufferSize	word	Size of Message Buffer
maxLogicalConnection	word	Maximum number of logical connections
maxBDataBlocks	word	Number of data blocks available simultaneously
maxBDataLen	word	Maximum size of a data block
pApplID	word*	16:16 (segmented) pointer to the location where Profile B should place the assigned application identification number

Return Value

Return Value	Comment
0x0000	Registration successful - application identification number has been assigned
All other values	Coded as described in parameter info class 0x10xx

7.4.2.2.1.2 CAPI_RELEASE

Description
The application PDD uses this operation to log off from Profile B.. Profile B shall release all resources that have been allocated for the application.

The application PDD is identified by the application identification number that had been assigned in the previous CAPI_REGISTER operation.

CAPI_RELEASE	**0x02**

Structure of command specific parameters:

Parameter	Type	Comment
ApplID	word	Application identification number that had been assigned by call of the function CAPI_REGISTER

Return Value

Return Value	Comment
0x0000	Release of the application successful
All other values	Coded as described in parameter 0x11xx

7.4.2.2.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application PDD transfers a message to Profile B. The application identifies itself with an application identification number. The pointer passed to Profile B is a 16:16 segmented pointer. The pointer in a DATA_B3_REQ message also is 16:16 segmented. The memory area of the message and the data block shall be either fixed or locked.

CAPI_PUT_MESSAGE	**0x03**

Structure of command specific parameters:

Parameter	Type	Comment
ApplID	word	Application identification number that had been assigned by call of the function CAPI_REGISTER
pCAPIMessage	void*	16:16 segmented pointer to the message that is passed to Profile B

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter 0x11xx

NOTE: When the process returns from the function call the message memory area can be reused by the application.

7.4.2.2.1.4 CAPI_GET_MESSAGE

Description
With this operation the application PDD retrieves a message from Profile B. The application PDD can only retrieve those messages intended for the stipulated application identification number. If there is no message waiting for retrieval, the function returns immediately with an error.

CAPI_GET_MESSAGE	**0x04**

Structure of command specific parameters:

Parameter	Type	Comment
ApplID	word	Application identification number that had been assigned by call of the function CAPI_REGISTER
ppCAPIMessage	void**	16:16 segmented pointer to the memory location where Profile B should place the 16:16 segmented pointer to the retrieved message

Return Value

Return Value	Comment
0x0000	Successful - Message was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx

NOTE: The received message may become invalid the next time the application issues a **CAPI_GET_MESSAGE** operation for the same application identification number.

7.4.2.2 Other functions

7.4.2.2.1 CAPI_SET_SIGNAL

Description
This operation is used by the application PDD to install a mechanism which signals the application PDD the availability of a message.

A call back mechanism is used between Profile B and an application PDD. By calling the IDC function with CAPI_SET_SIGNAL function code the application PDD passes a 16:16 (segmented) pointer to a call back function to Profile B.

CAPI_SET_SIGNAL	**0x05**

Structure of command specific parameters:

Parameter	Type	Comment
ApplID	word	Application identification number that had been assigned by call of the function CAPI_REGISTER
signFunc	void*	16:16 segmented pointer to the call-back function

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

NOTE: The call-back function is called by Profile B**,** after
any message is queued in application's message queue
a notified busy condition changed
a notified queue full condition changed
Interrupts are disabled. The call-back function shall be terminated via RETF. All registers shall be preserved. At the time of calling, at least 32 bytes are available on the stack.
The call-back function shall be called with interrupts disabled. Profile B shall not call this function recursively, even if the call-back function enables interrupts. Instead the call-back function shall be called again after returning to Profile B.
The call-back function is allowed to use Profile B operations **CAPI_PUT_MESSAGE**, **CAPI_GET_MESSAGE**, and **CAPI_SET_SIGNAL**. In that case the call-back function shall be aware that interrupts may be enabled by Profile B.
In case of local confirmations (e.g. LISTEN_CONF) the call-back function may be activated before the operation CAPI_PUT_MESSAGE returns to the application.

7.4.2.2.2.2 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B (DLL). SzBuffer on call is a 16:16 (segmented) pointer to a buffer of 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

Function call

CAPI_GET_MANUFACTURER	0x06

Structure of command specific parameters:

Parameter	Type	Comment
SzBuffer	char*	16:16 (segmented) pointer to a buffer of 64 bytes

Return Value

Return	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

7.4.2.2.2.3 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

Function call

CAPI_GET_VERSION	0x07

Structure of command specific parameters:

Parameter	Type	Comment
pCAPIMajor	word*	16:16 (segmented) protected mode pointer to a word receiving Profile B major version number: 2
pCAPIMinor	word*	16:16 (segmented) protected mode pointer to a word receiving Profile B minor version number: 0
pManufacturerMajor	word*	16:16 (segmented) protected mode pointer to a word receiving manufacturer specific major number
pManufacturerMinor	word*	16:16 (segmented) protected mode pointer to a word receiving manufacturer specific minor number

Return Value

Return	Comment
0x0000	No error, version numbers are copied
All other values	Coded as described in parameter info class 0x11xx

7.4.2.2.2.4 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. SzBuffer on call is a 16:16 (segmented) protected mode pointer to a buffer of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

Function call

CAPI_GET_SERIAL_NUMBER	0x08

Structure of command specific parameters:

Parameter	Type	Comment
SzBuffer	char*	16:16 (segmented) pointer to a buffer of 8 bytes

Return Value

Return	Comment
0x0000	No error SzBuffer contains the serial number in plain text in the form of a 7-digit number. If no serial number is provided by the manufacturer, an empty string is returned.
All other values	Coded as described in parameter info class 0x11xx

7.4.2.2.2.5 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. SzBuffer on call is a 16:16 (segmented) protected mode pointer to a buffer of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. *CtrlNr* contains the controller number (bit 0..6), for which this information is requested.

CAPI_GET_PROFILE	**0x09**

Structure of command specific parameters:

Parameter	Type	Comment
SzBuffer	void*	16:16 (segmented) protected mode pointer to a buffer of 64 bytes
CtrlNr	word	Number of Controller. If 0, only number of installed controllers is given to the application.

Return Value

Return	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
word	number of installed controllers, least significant octet first
word	number of supported B-channels, least significant octet first
dword	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4.[31]: reserved
dword	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
dword	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
dword	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for fax group 3 5..31: reserved
6 dwords	reserved for Profile B usage
5 dwords	manufacturer specific information
NOTE: This function can be extended, so an application has to ignore unknown bits. Profile B shall set every reserved field to 0.	

7.5 Novell NetWare

7.5.1 NetWare Operation System specific implementation for Profile A

7.5.1.1 Introduction

This implementation is supported by system NetWare V3.12 or higher.

There are two mechanisms to support the Profile A exchange method under the NetWare system. The functions PciGetProperty and PciRegister are dynamically linked to the NAF when called. The functions PciDeregister, PciPutMessage, PciGetMessage and PciSetSignal are directly provided by the NAF when the PUF registers.

Every NAF shall be NLM and shall export an entry point per Profile A function which are dynamically linked. Names of those functions are different for each NAF and should be unique in a system. They are built by concatenating the NAF name (the NLM file name) with the PCI functions names. For example, for a NAF called NAF.NLM, the exported functions are called NAF_PciGetProperty and NAF_PciRegister. The other functions (PciDeregister, PciPutMessage, PciGetMessage and PciSetSignal) should not be exported. PciGetHandles is only an interface function and is not provided at all by a NAF. This function needs to access to a Btrieve database called PCI.BTV and is located in the directory SYS:\PCI\.

PciRegister and PciGetProperty check if the functions, whose names are in the database, are actually loaded in the system. To access to a NAF, a PUF shall know the NAF's name, as entry key in the database. The NAF's name is a logical name attributed by the user at load time (on the load command line for example). The addresses access to the NLM may be provided transparently to the PUF inside the Pci's exchange mechanism functions as described in the annex J.

The NAF is loaded at initialisation time (or later, but always before the PUF). Its name is unique, and is used as a key index in the database.

When a PUF registers, the NAF provides pointers to the non-exported PCI functions. The PUF needs to keep trace of these pointers. So they are parts of the Exchange Identifier. The NAF also needs to keep trace of the PUF, so it assigns an Identifier to the PUF at registration time, this NAF-provided Identifier is another part of the Exchange Identifier.

The "C" calling convention is used by the Profile A exchange method under NetWare.

In NetWare, the memory model used is the flat model. So every pointers are 32 bits flat pointers.

The structures alignment is **byte**.

7.5.1.2 Mapping of generic types and constants

In NetWare, the following data types are used:

BYTE	unsigned 8 bits integer value
WORD	unsigned 16 bits integer value
LONG	unsigned 32 bits integer value

The following constants are defined:

```
PCI_HANDLE_SIZE             19
PCI_DATABASE_NAME           "SYS:\PCI\PCI.BTV"
PCI_DATABASE_PATH           "SYS:\PCI\"

/*
 * Basic types
 */
typedef LONG                PCI_INTEGER
typedef BYTE                * PCI_BYTEARRAY
typedef BYTE                PCI_HANDLE[PCI_HANDLE_SIZE];
typedef void                        (* PCI_PROCEDURE)();
typedef struct {
        PCI_INTEGER         ExchangeId;             /* Unique Identifier provided by NAF to identify the PUF */
        PCI_INTEGER         (* PciDeregisterPtr)(); /* Address of the PciDeregister function */
        PCI_INTEGER         (* PciPutMessagePtr)(); /* Address of the PciPutMessage function */
        PCI_INTEGER         (* PciGetMessagePtr)(); /* Address of the PciGetMessage function */
        PCI_INTEGER         (* PciSetSignalPtr)();  /* Address of the PciSetSignal function */
} PCI_EXID;

/*
 * Structures
 */
struct pci_mpb {
        PCI_INTEGER         MessageID;
        PCI_INTEGER         MessageMaximumSize;
        PCI_INTEGER         MessageActualUsedSize;
        PCI_INTEGER         DataMaximumSize;
        PCI_INTEGER         DataActualUsedSize;
};
typedef struct pci_mpb PCI_MPB;
```

```
struct pci_register {
        PCI_INTEGER     PUFVersion;
        PCI_INTEGER     PUFType;
        PCI_INTEGER     MaxMsgSize;
};

/*
 * Exchange functions prototypes
 */
PCI_INTEGER PciGetHandles(      PCI_INTEGER MaxHandles,
                                PCI_BYTEARRAY PCIHandles,
                                PCI_INTEGER * ActualHandles);

PCI_INTEGER PciGetProperty(     PCI_HANDLE PCIHandle,
                                PCI_INTEGER MaximumSize,
                                PCI_BYTEARRAY Property,
                                PCI_INTEGER * ActualSize);

PCI_INTEGER PciRegister(        PCI_HANDLE PCIHandle,
                                struct pci_register * PCIRegisterInfo,
                                PCI_EXID * ExID);

PCI_INTEGER PciDeregister(      PCI_EXID * ExID);

PCI_INTEGER PciPutMessage(      PCI_EXID * ExID,
                                PCI_MPB * PCIMPB,
                                PCI_BYTEARRAY Message,
                                PCI_BYTEARRAY Data);

PCI_INTEGER PciGetMessage(      PCI_EXID * ExID,
                                PCI_MPB * PCIMPB,
                                PCI_BYTEARRAY Message,
                                PCI_BYTEARRAY Data);

PCI_INTEGER PciSetSignal(       PCI_EXID * ExID,
                                PCI_INTEGER Signal,
                                PCI_PROCEDURE SignalProcedure);
```

7.5.1.3 Description of functions

This subclause describes the implementation, under NetWare, of the Profile A exchange method functions. During a PUF to NAF call, the size of the stack should be at least 1 024 bytes deep.

7.5.1.3.1 PciGetHandles

The PciGetHandles uses a Btrieve database called PCI.BTV, located in the directory SYS:\PCI\. Each record of the database corresponds to an installed NAF. Each record has the format:

Field	Format	Size	Position	Key index
NAF's name	zero terminated string	19	1	yes
NAF's driver name	zero terminated string	9	20	no

The size of a record is 28 bytes.

The NAF's driver name is used to build the name of the NAF's exported functions, as it is explained in subclause 7.5.1.1.

The following operations shall get all names of installed NAF drivers:

⇒ Open the Database
⇒ Get the first record
⇒ while the end of the database is not reached
⇒ Get next record
⇒ Close the database.

7.5.1.3.2 PciGetProperty

This function is in charge of providing to the PUF the PROPERTY of the NAF. Implicitly, it checks if the NAF is available, by linking to NAF's PciGetProperty function via the ImportSymbol function.

The following operations shall take place in order:

⇒ Open the database
⇒ Get the record corresponding to the NAF's name
⇒ Close the database
⇒ Get the NAF's PciGetProperty function's name
⇒ Link to this function (get its address)
⇒ Call to this address with the parameters provided by the PUF.

7.5.1.3.3 PciRegister

This function is in charge of providing an association between a PUF and a NAF. The NAF is loaded and the PUF is linked with the NAF's exported functions. The availability of the chosen NAF is checked during the link to its functions.

While at least one PUF is registered to a NAF, this NAF should not be unloaded. So, the NAF must provide to the system a NLM NetWare CHECK function to inform the operator when it is in service and cannot be unloaded. As soon as the last registered PUF deregisters, the NAF can be unloaded again. On NetWare V4.0 or higher, the use of functions *SetNLMDontUnloadFlag* and *ClearNLMDontUnloadFlag* to prevent NAF to be unloaded while PUF are registered is highly recommended.

The following operations shall take place in order:

⇒ Open database
⇒ Get the record corresponding to the NAF's name
⇒ Close the database
⇒ Get the NAF's PciRegister function's name
⇒ Link to this function (get its address)
⇒ Call to this address with the parameters provided by the PUF
⇒ on return from the NAF, the Exchange Identifier's fields and the maximum message size parameter of the registration structure have been provided by the NAF
⇒ return to the PUF with the return code from the NAF.

7.5.1.3.4 PciDeregister

This function is charged of disassociating a PUF and a NAF. When all the PUF registered have deregister, the NAF could allow to be unloaded.

The following operations should take place, in order:

⇒ Get the address of the NAF's PciDeregister function in the Exchange Identifier
⇒ Call the address of the PciDeregister function stored in the Exchange Identifier to inform the NAF of the end of the association.

7.5.1.3.5 PciPutMessage

This function is in charge of providing a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as indicated in the generic description of the PciPutMessage function. Either Message or Data may be optional. In this case they are specified as NULL.

The following operations shall take place, in order:

⇒ Get the address of the NAF's PciPutMessage function in the Exchange Identifier

⇒ Call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.5.1.3.6 PciGetMessage

This function is in charge of providing a PUF with a message, and associated data if any, from a NAF. Parameters are provided in the same order as indicated in the generic description of the PciGetMessage function. The NAF write the return data directly in the buffers provided by the PUF. Either Message or Data may be optional. In this case they are specified as NULL.

The following operations shall take place, in order:

⇒ Get the address of the NAF's PciPutMessage function in the Exchange Identifier

⇒ Call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.5.1.3.7 PciSetSignal

This function allows a PUF of providing a direct information mechanism to be used by the NAF in case of incoming event. Two mutually exclusive mechanisms are offered under NetWare:

– a local semaphore mechanism;

– a callback function mechanism.

Once a mechanism is chosen by the PUF, the other is de-activated by the NAF for that particular PUF. Both method shall be supported by the NAF.

The first mechanism does not use the SignalProcedure parameter. This parameter shall be set to 0.

The second mechanism used the SignalProcedure to identify the function to be called by the NAF. In that case, the SignalProcedure parameter shall not be equal to 0.

The following operations shall take place, in order:

⇒ Get the address of the NAF's PciSetSignal function in the Exchange Identifier;

⇒ Call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.5.1.3.7.1 Local semaphore mechanism

The local semaphore mechanism uses a CLIB local semaphore of the NetWare system. PUF which maintain a CLIB process context should select this mechanism. The PUF process can then wait on the local semaphore. When an inbound message is available, the NAF will signal the local semaphore causing the PUF to make up and retrieve a message, by calling the PciGetMessage function.

The PUF is responsible for giving an initialized local semaphore to the NAF in the Signal parameter when calling the function PciSetSignal. After deregistering or deactivating the signal mechanism, the PUF shall also close the local semaphore.

7.5.1.3.7.2 Callback function mechanism

PUF which do not maintain a CLIB process context should use the call-back function mechanism. The PUF supply a pointer to a PUF resident notification function in the SignalProcedure parameter and a PUF defined context value in the Signal parameter. Each time an inbound message is available, the NAF call the PUF's notification function with the PUF context value in parameter. The PUF shall then call the PciGetMessage function to retrieve the available message.

The call-back function can be called from either the process or interrupt context. So blocking operations, such as disk input output should not be performed by the signal procedure. If blocking operations are required, they should be executed from a separate application supplied process. For the same reasons, a call-back function should not call the NAF directly.

7.5.1.3.7.3 De-activation mechanism

To deactivate any signal mechanism the PciSetSignal function Signal and SignalProcedure parameters shall be NULL. Once deactivated, the previous mechanism shall no longer be used by the NAF to call the PUF.

7.5.1.4 Availability of NAFs

In order to ensure consistence of the database and to export in an unambiguous way its functions, NAF should be re-entrant if it can be loaded multiple times.

7.5.1.4.1 Declaration action

In case of a NAF that can be loaded multiple times, it should check that it is the first time that it is loaded with that name. If not, it should not load.

When loading, the NAF shall declare itself in the database. It may first get the list of available PCI_HANDLES to check if not already declared. The mechanism the NAF uses is the same as any PUF to get available NAF: PciGetHandles. If it is already declare, it shall check that the configuration is correct. Otherwise, it should call the PciGetProperty function of the NAF described in the database to determine if this NAF is actually active. If so, the NAF should not load. If not, the record is deleted.

In the case the database does not exist, the NAF should create it (as it should create the directory SYS:\PCI if it does not exist).

If not yet declare, or if the declaration was incorrect, the NAF declares itself by inserting a new record in the database.

There is no theoretical limit to the number of NAF that can register in the database.

7.5.1.4.2 Extraction action

When the NAF is unloaded, it should extract its declaration from the database. For doing so, it should delete the record in the database during the unloading procedure.

7.5.2 NetWare for Profile B

The NetWare server operating system provides an open, non-preemptive, multitasking platform including file, print, communications and other services. A typical NetWare server can support tens to hundreds of simultaneous users. Extensibility of communication services in particular is accommodated through open service interfaces allowing integration of third party hardware and software. Therefore when considering the addition of a new communications subsystem to the NetWare operating system, scalability and flexibility are considered primary design goals.

This implementation of Profile B in the NetWare server operating system addresses both scalability and flexibility by allowing concurrent operation of multiple Profile B compliant applications and multiple ISDN controllers provided by different manufacturers. Profile B service provider in the NetWare operating system environment is a subset of the overall NetWare CAPI Manager subsystem. The NetWare CAPI Manager includes all standard functions defined by Profile B as well as auxiliary functions providing enhanced ISDN resource management for NetWare systems running multiple concurrent Profile B applications. The NetWare CAPI Manager subsystem also includes a secondary service interface which integrates each manufacturer specific ISDN controller driver below Profile B. Although the driver interface maintains the general structure and syntax of Profile B functions and messages, it is not part of Profile B definition. The driver interface is unique to the NetWare CAPI Manager implementation.

The following description of Profile B within the NetWare server operating system provides a detailed description of each standard Profile B function which makes up the application programming interface, containing sufficient information to implement Profile B compliant applications within the NetWare environment. A general overview of the NetWare CAPI Manager is also provided to identify which services are standard Profile B and which are unique to the NetWare CAPI Manager subsystem. Detailed description of the NetWare CAPI Manager unique functions for enhanced resource management and ISDN controller software integration is beyond the scope of this document. The complete definition is contained in the Novell specification **NetWare CAPI Manager and CAPI Driver specification** (Version 2.0).

Architectural overview

The NetWare CAPI Manager, which is implemented as a NetWare Loadable Module (NLM) acts as a service multiplexer and common interface point between Profile B compliant applications and each manufacturer specific ISDN controller driver residing below Profile B. Each Profile B application and each controller driver is implemented as a separate NLM which independently registers with the NetWare CAPI Manager at initialization time. Profile B exists between the Profile B applications and the NetWare CAPI Manager. NetWare CAPI Manager auxiliary management functions also exits at this point. A Novell defined service interface exists between the NetWare CAPI Manager and the ISDN controller drivers however applications have no knowledge of this lower level interface. From the application perspective, the lower level driver interface is an internal detail of the NetWare CAPI Manager implementation of Profile B.

Figure 37 illustrates the relationship between Profile B applications, the NetWare CAPI Manager, and manufacturer specific controller drivers and controller hardware.

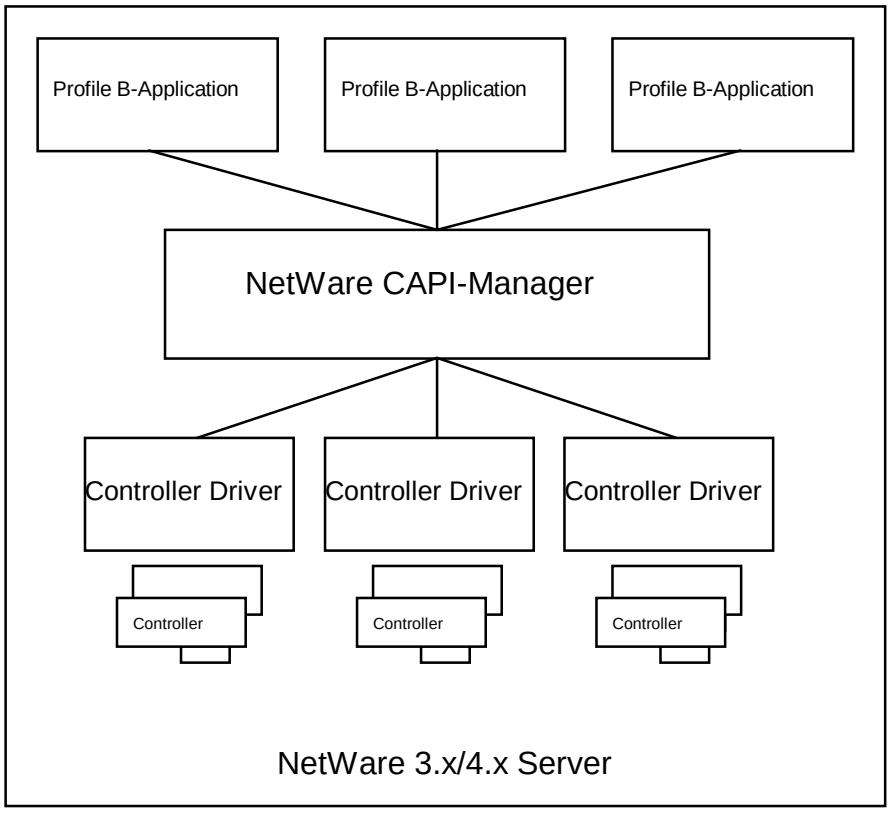


Figure 37: Architectural overview

Services provided by the NetWare CAPI Manager are presented as a set of exported public symbols. To avoid public symbol conflicts within the server environment, services provided by each controller driver are presented as a set of entry point addresses supplied to the NetWare CAPI Manager at driver registration time. NetWare CAPI Manager services include the standard Profile B function set, auxiliary functions supporting driver registration and deregistration of controller services and auxiliary management functions referenced by Profile B applications.

The additional management functions implement a powerful search mechanism for locating specific controller resources and a locking mechanism to reserve controller resources for exclusive use by an application. The CAPI_GetFirstCntlrInfo searches for the first occurrence of a controller whose capabilities match the search criteria specified by the application. The search criteria can include a symbolic controller name, specific protocols, required bandwidth, etc. The CAPI_GetNextCntlrInfo function searches for additional controllers which meet the previously specified search criteria. The CAPI_LockResource function is provided for applications which shall have guaranteed access to a previously identified controller channel or protocol resources. The specified resource remains reserved until the application calls the CAPI_FreeResource function. These additional management functions are intended to provide enhanced management capabilities in server systems configured with a variety of controllers or a large number of concurrently executing applications.

To insure efficient operation of multiple applications and drivers in the server environment, inbound message signalling is required by the NetWare CAPI Manager. The CAPI_Register function defines additional signal parameters which shall be provided by the application to successfully register. Applications are not permitted to poll for inbound messages. Because signalling is required and signal parameters are specified at registration time, the CAPI_SetSignal function is not included in this implementation of Profile B.

Refer to the **NetWare CAPI Manager & CAPI Driver Specification** for a complete definition of the auxiliary and driver functions. The function descriptions provided in this section reflect only the standard Profile B function set provided by the NetWare CAPI Manager. Note that in some cases the parameter lists required by the NetWare CAPI Manager version of Profile B functions are different from other operating system implementations.

Function Call Conventions in NetWare environment:

- all interface functions conform to standard "C" language calling conventions;

- all functions can be called from either a process or interrupt context.

- profile B defines a standard 16 bit error code format where bits 8 - 15 identify the error class and bits 0 - 7 identify the specific error. With one exception, this approach is used throughout this specification. The exception is that all functions return either a DWORD (unsigned long) or a void type rather than a 16 bit WORD type. Bits 31 - 16 of the return value shall always be zero.

Data Type Conventions in NetWare environment:

- structures were used with byte alignment;
- the following additional simple data types were used:

BYTE	unsigned 8 bit integer value
WORD	unsigned 16 bit integer value
DWORD	unsigned 32 bit integer value
BYTE *	32 bit pointer to an unsigned char
WORD *	32 bit pointer to an unsigned 16 bit integer
VOID *	32 bit pointer
VOID **	32 bit pointer to a 32 bit pointer

7.5.2.1 Message operations

7.5.2.1.1 CAPI_Register

Description
Applications use *CAPI_Register* to register their presence with Profile B. Registration parameters specify the maximum number of ISDN logical connections, message buffer size, number of data buffers and data buffer size required by the application. Message buffer size is normally calculated according to following formula:

Message buffer size = 1 024 + (1 024 * number of ISDN logical connections)

Inbound message signalling parameters are also supplied. Successful registration causes Profile B to assign a system unique application identifier to the caller. The application identifier is used in subsequent Profile B function calls as well as in Profile B defined messages. Two inbound message availability signalling options are supported. The signalType and signalHandle parameters allow an application to select either CLIB Local Semaphore or direct function call-back notification. Application polling of the inbound message queue shall not be permitted. Successful application registration requires selection of an inbound message signalling mechanism.

Applications which maintain a CLIB process context should select Local Semaphore signalling via the signalType parameter and supply a previously allocated Local Semaphore handle as the signalHandle parameter. The application receive process can then wait on the local semaphore. When an inbound message is available, the Profile B driver shall signal the local semaphore causing the application process to wakeup and retrieve a message, by calling the *CAPI_GetMessage* function.

Applications which do not maintain a CLIB process context should select direct call-back signalling via the signalType parameter, supply a pointer to an application resident notification function as the signalHandle parameter and an application defined context value as the signalContext parameter. When an inbound message is available, Profile B shall call the specified application notification function, supplying the application context value. The application shall call the *CAPI_GetMessage* function to retrieve any available messages.

Function call

```
DWORD CAPI_Register(    WORD messageBufSize,
                        WORD connectionCnt,
                        WORD dataBlockCnt,
                        WORD dataBlockLen,
                        WORD *applicationID
                        WORD signalType,
                        DWORD signalHandle,
                        DWORD signalContext,
                        );
```

Parameter	Comment
messageBufSize	Specifies the message buffer size.
connectionCnt	Specifies the maximum number of logical connections this application can concurrently maintain. Any application attempt to exceed the logical connection count by accepting or initiating additional connections shall result in a connection establishment failure and an error indication from the Profile B driver.
dataBlockCnt	Specifies the maximum number of received data blocks that can be reported to the application simultaneously for each B-channel connection. The number B-channel data blocks has a decisive effect on the throughput of B-channel data in the system and should be between 2 and 7. At least two B-channel data block shall be specified.
dataBlockLen	Specifies maximum size of a B-channel data unit which can be transmitted and received. Selection of a protocol that requires larger data units and attempts to transmit or receive larger data units shall result in an error from Profile B.
applicationID	This parameter specifies a pointer to a location where the Netware CAPI Manager shall place the assigned application identifier during registration. This value is valid only if the registration operation was successful, as indicated by a return code of 0x0000.
signalType	Specifies the inbound message signalling mechanism selected by the application. The signalling mechanism is used by the driver to notify the application when inbound control or data messages are available or when queue full / busy conditions change. The signalType parameter also defines the meaning of the signalHandle parameter. Two signalType constants are defined as follows: 0x0001 SIGNAL_TYPE_LOCAL_SEMAPHORE 0x0002 SIGNAL_TYPE_CALLBACK.
signalHandle	Depending on the value of the signalType parameter, signalHandle specifies either the local semaphore handle previously allocated by the application or the address of an application resident receive notification function with the following format: void CAPI_ReceiveNotify(DWORD signalContext); (see below).
signalContext	If the signalType parameter contains SIGNAL_TYPE_CALLBACK, the signalContext specifies an application defined context value. This value shall be passed to the application notification function. The signalContext value has no meaning to the Profile B. It may be used by an application to reference internal data structures etc during receive notification callback process. If the signalType parameter specifies SIGNAL_TYPE_LOCAL_SEMAPHORE this value is ignored.

Return Value

Return Value	Comment
0x0000	Registration successful - application identification number has been assigned
All other values	Coded as described in parameter info class 0x10xx

7.5.2.1.2 CAPI_ReceiveNotify

CAPI_ReceiveNotifyDescription

This optional application resident receive notification function is called by the NetWare CAPI Manager implementation of the Profile B whenever an inbound message addressed to the application is available. This function is intended for exclusive use by NetWare system applications which do not maintain a CLIB context. Use of this function is enabled at application registration time by specifying the CAPI_Register signalType parameter as SIGNAL_TYPE_CALLBACK. Note that non-system level applications should always use local semaphores for receive message notification by specifying the CAPI_Register signalType parameter as SIGNAL_TYPE_LOCAL_SEMAPHORE.

Each time the CAPI_ReceiveNotify function is called, it should in turn call the CAPI_GetMessage to retrieve the next available message addressed to the application. The signalContext parameter passed to the CAPI_ReceiveNotify function contains an application defined context value previously supplied to the CAPI_Register function. This value is meaningful only to the application, for example as an internal data structure pointer

NOTE: The CAPI_ReceiveNotify function can be called from either the process or interrupt context. To avoid adverse system impact, blocking operations such as disk input output should not performed by the receive notify function. If blocking operations are required they should be executed from a separate application supplied process.

7.5.2.1.3 CAPI_Release

Description

Applications uses *CAPI_Release* to deregister from Profile B. All memory allocated on behalf of the application by Profile B shall be released.

Function call

```
DWORD CAPI_Release (WORD ApplID);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_Register

Return Value

Return Value	Comment
0x0000	Release of the application successful
All other values	Coded as described in parameter info class 0x11xx

7.5.2.1.4 CAPI_PutMessage

Description

Applications call *CAPI_PutMessage* to transfer a single message to Profile B.

Function call

```
DWORD CAPI_PutMessage(    WORD ApplID,
                          VOID *pCAPIMessage
                          );
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_Register
pCAPIMessage	Points to a memory block which contains a message for the Profile B Driver

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

NOTE: When the process returns from the function call the message memory area can be reused by the application.

Description

Applications call *CAPI_GetMessage* to retrieve a single message from Profile B. If a message is available, it address is returned to the application in location specified by the *ppCAPIMessage* parameter. If there are no messages available from any of the registered drivers, *CAPI_GetMessage* returns with an error indication

The contents of the message blocks returned by this function is valid until the same application calls *CAPI_GetMessage* again. In cases where the application will process the message asynchronously or needs to maintain the message beyond the next call to *CAPI_GetMessage,* a local copy of the message shall be made by the application.

Function call

```
DWORD CAPI_GetMessage(      WORD ApplID,
                            VOID** ppCAPIMessage);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_Register
ppCAPIMessage	Pointer to the memory location where the Netware CAPI Manager should place the retrieved message address. The contents of the output variable specified by msgPtr is valid only if the return code indicates no error

Return Value

Return Value	Comment
0x0000	Successful - Message was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx

7.5.2.2 Other functions

7.5.2.2.1 CAPI_GetManufacturer

Description

Applications call *CAPI_GetManufacturer* to retrieve manufacturer specific identification information from the specified ISDN controller.

Function call

```
DWORD CAPI_GetManufacturer(     DWORD Controller,
                                BYTE *szBuffer
                                );
```

Parameter	Comment
Controller	Specifies the system unique controller number for which the manufacturer information is to be retrieved. Coding is described in clause 6.
szBuffer	Specifies a pointer to an application data area 64 bytes long which will contain the manufacturer identification information upon successful return. The identification information is represented as a zero terminated ASCII text string.

Return Value

Return Value	Comment
0x0000	Successful - information was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx

7.5.2.2.2 CAPI_GetVersion

Description

Applications call *CAPI_GetVersion* to retrieve version information from the specified ISDN controller. Major and minor version numbers are returned for both Profile B and the manufacturer specific implementation.

Function call

```
DWORD CAPI_GetVersion(    DWORD Controller,
                          WORD* pCAPIMajor,
                          WORD* pCAPIMinor,
                          WORD* pManufacturerMajor,
                          WORD* pManufacturerMinor
                          WORD *pManagerMajor
                          WORD *pManagerMinor
                          );
```

Parameter	Comment
Controller	Specifies the system unique controller number for which the manufacturer information is to be retrieved. Coding is described in clause 6.
pCAPIMajor	Pointer to a WORD receiving Profile B major version number: 0x0002
pCAPIMinor	Pointer to a WORD receiving Profile B minor version number: 0x0000
pManufacturerMajor	Pointer to a WORD receiving manufacturer specific major number
pManufacturerMinor	Pointer to a WORD receiving manufacturer specific minor number
pManagerMajor	Pointer to a WORD receiving Netware CAPI Manager major version number
pManagerMinor	Pointer to a WORD receiving Netware CAPI Manager minor version number

Return Value

Return	Comment
0x0000	No error, version numbers are copied
All other values	Coded as described in parameter info class 0x11xx

7.5.2.2.3 CAPI_GetSerialNumber

Description

Applications call *CAPI_GetSerialNumber* to retrieve the optional serial number of the specified ISDN controller.

Function call

```
DWORD CAPI_GetSerialNumber(    DWORD Controller,
                               BYTE *szBuffer
                               );
```

Parameter	Comment
Controller	Specifies the system unique controller number for which the serial number information is to be retrieved. Coding is described in clause 6.
szBuffer	Pointer to a buffer of 8 bytes

Return Value

Return	Comment
0x0000	No error szBuffer contains the serial number in plain text in the form of a 7-digit number. If no serial number is provided by the manufacturer, an empty string is returned.
All other values	Coded as described in parameter info class 0x11xx

7.5.2.2.4 CAPI_GetProfile

Description

The application uses this function to get the capabilities from Profile B. *Buffer* on call is a pointer to a buffer of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. *Controller* contains the controller number (bit 0..6), for which this information is requested.

```
DWORD CAPI_GetProfile (     VOID *Buffer,
                            DWORD Controller
                            );
```

Parameter	Comment
Buffer	Pointer to a buffer of 64 bytes
Controller	Number of Controller. If 0, only number of installed controllers is given to the application.

Return Value

Return	Comment
0x0000	No error Buffer contains the requested information.
All other values	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
WORD	Number of installed controllers, least significant octet first
WORD	Number of supported B-channels, least significant octet first
DWORD	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4..31: reserved
DWORD	B1 protocols support (bit field): 0: 64 kbit/s with HDLC framing, always set. 1: 64 kbit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kbit/s inverted with HDLC framing. 6: 56 kbit/s bit transparent operation with byte framing from the network 7..31: reserved
DWORD	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
DWORD	B3 protocol support (bit field): 0: Transparent, always set 1: T.90 NL with compatibility to T.70 NL according to T.90 [16] Appendix II. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
6 DWORDs	reserved for Profile B usage
5 DWORDs	manufacturer specific information
NOTE: This function can be extended, so an application shall ignore unknown bits. Profile B shall set every reserved field to 0.	

7.6 Windows/NT

7.6.1 Windows NT operation system specific implementation for Profile A

7.6.1.1 Introduction

This implementation is supported starting from release 3.1 of Windows NT.

This specification describes two separate interfaces between NAF and PUF. The first one describes a DLL mechanism (where NAF is a Dynamic Link Library). In the other one, the NAF is a Device Driver running in Kernel under Windows NT.

The PUF part can run under Windows NT User mode or Kernel mode.

7.6.1.1.1 DLL version

Except for the PciGetHandles function call, calls of functions dynamically linked is the basic mechanism used to support the Profile A exchange method under Windows NT.

Every NAF shall be a DLL (Dynamic Linked Library) and shall export an entry point per Profile A function.

Exported functions provided by a NAF are **PciGetProperty**, **PciRegister**, **PciGetMessage**, **PciPutMessage**, **PciSetSignal**, **PciDeregister**.

To access a NAF the only requirement for the PUF is to know the name of the corresponding DLL.

The PciRegister function dynamically loads the NAF. It needs to keep trace of the handle of the NAF as a DLL, so this handle is part of the Exchange Identifier. The NAF also needs to keep trace of the PUF, so it assigns an identifier to the PUF at registration time. This NAF-provided identifier is the other part of the Exchange Identifier.

The functioning model under Windows NT is 32 bits using a flat-addressing scheme. So all pointers are 32 bits flat pointers.

7.6.1.1.2 Device driver version

Under Windows NT a device driver can be called from user mode with the **DeviceIoControl** function or from kernel mode with IDC mechanism. Io control code and input buffer are described for user and kernel modes. Figure 38 presents the Windows/NT general architecture.

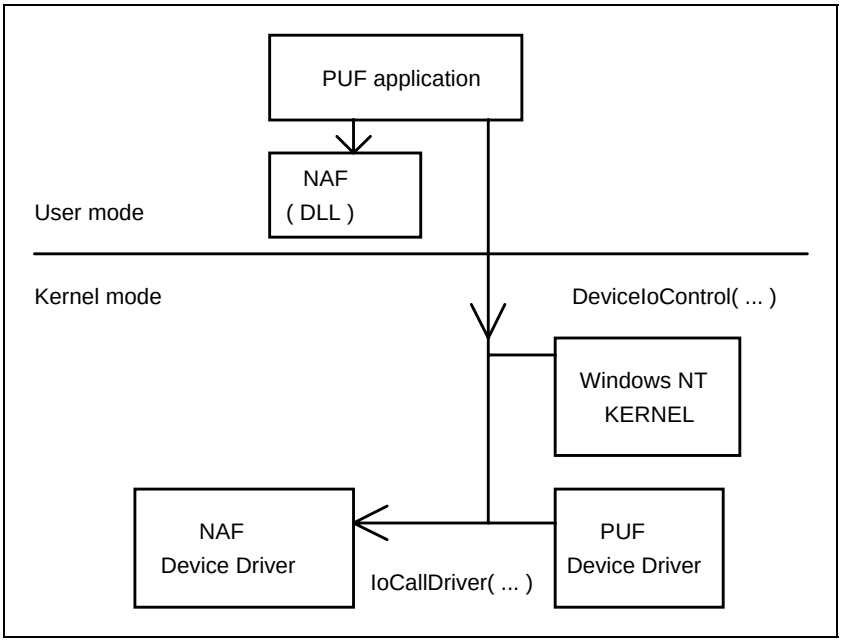


Figure 38: Windows/NT general architecture

7.6.1.1.3 Driver access method from user mode

The **CreateFile** function is used to get a driver handle. At the end of the communication session the handle is released by **CloseHandle** function.

User applications call PCI NAF device driver via the **DeviceIoControl** function.

The Windows NT registry supplies a list of NAF device drivers.

An event can be used to synchronise PciGetMessage on event, PUF pass an existing event by means of **lpOverlapped** parameter of the **DeviceIoControl** function.

The caller (PUF) creates an event and passes it to the PCI device driver through the **OVERLAPPED** structure. This method is used to map the PciSetSignal event service.

The callback mechanism is not available from a kernel device driver to a user process.

7.6.1.1.4 Driver access method from kernel mode

The mechanism to call a device driver from another one is to get an object on the device driver with the **IoGetDeviceObjectPointer** Windows NT kernel function; build an IRP (I/O Request Packet) with **IoBuildDeviceIoControlRequest** and then, call the device driver with **IoCallDriver** function.

The IRP input buffer format is the same as the User/Kernel specification.

The NAF device driver receives the PUF request in the IRP_MJ_DEVICE_CONTROL MajorFunction registered in the DeviceEntry function, and get the NAF requested service with the IoControlCode.

An event can be passed to the NAF through the IRP. PUF device driver can wait on event after calling the PUF with IoCallDriver function. PUF can create a kernel thread and use the PciSetSignal service.

NAF device driver sample code

```
pIrpStack = IoGetCurrentIrpStackLocation(pIrp);
// Dispatch based on major fcn code.
switch (pIrpStack->MajorFunction)
{
        case IRP_MJ_DEVICE_CONTROL:                     //  Dispatch on IOCTL PUF requests
                switch (pIrpStack->Parameters.DeviceIoControl.IoControlCode)
                {
                        case IOCTL_PCIGETPROPERTY:  break;
                        case IOCTL_PCIREGISTER:     break;
                        case IOCTL_PCIDEREGISTER:   break;
                        case IOCTL_PCIPUTMESSAGE:   break;
                        case IOCTL_PCIGETMESSAGE:   break;
                        case IOCTL_PCISETSIGNAL:    break;
                }
                break;
}
```

As we use METHOD_BUFFERED, NT copies input buffer in pIrp->AssociatedIrp.SystemBuffer before entry and copies it to output buffer after return.

7.6.1.2 Mapping of generic types and constants

Under Windows NT, the PCI_HANDLE is the name of the NAF's DLL.

The structure alignment is **byte**.

```
/*
 * Basic data types
 */
typedef SHORT PCI_INTEGER;
typedef LPSTR PCI_BYTEARRAY;
typedef LPSTR PCI_HANDLE;
typedef void (*PCI_PROCEDURE)();

typedef struct {
        HANDLE          DLLHandle;// NAF's DLL Handle
        PCI_INTEGER     Exchange_Id;        // PUF's identifier
} PCI_EXID;
```

```
/*
 * Structures definition
 */
struct pci_mpb {
        PCI_INTEGER         MessageID;
        PCI_INTEGER         MessageMaximumSize;
        PCI_INTEGER         MessageActualUsedSize;
        PCI_INTEGER         DataMaximumSize;
        PCI_INTEGER         DataActualUsedSize;
};
typedef struct pci_mpb PCIMPB;

struct pci_register {
        PCI_INTEGER         PUFVersion;
        PCI_INTEGER         PUFType;
        PCI_INTEGER         MaxMsgSize;
};

/*
 * Exchange functions prototypes
 */
PCI_INTEGER PciGetHandles(      PCI_INTEGER MaxHandles,
                                PCI_BYTEARRAY PCIHandles,
                                PCI_INTEGER *ActualHandles);

PCI_INTEGER PciGetProperty(     PCI_HANDLE PCIHandle,
                                PCI_INTEGER MaximumSize,
                                PCI_BYTEARRAY Property,
                                PCI_INTEGER *ActualSize);

PCI_INTEGER PciRegister(        PCI_HANDLE PCIHandle,
                                struct pci_register * PCIRegisterInfo,
                                PCI_EXID *ExID);

PCI_INTEGER PciDeregister(      PCI_EXID *ExID);

PCI_INTEGER PciPutMessage(      PCI_EXID *ExID,
                                PCI_MPB *PCIMPB,
                                PCI_BYTEARRAY Message,
                                PCI_BYTEARRAY Data);

PCI_INTEGER PciGetMessage(      PCI_EXID *ExID,
                                PCI_MPB *PCIMPB,
                                PCI_BYTEARRAY Message,
                                PCI_BYTEARRAY Data);

PCI_INTEGER PciSetSignal(       PCI_EXID *ExID,
                                PCI_INTEGER Signal,
                                PCI_PROCEDURE SignalProcedure);
```

Device driver structures:

```
#define COMMON_MAX_SIZE     4096

struct IoPciGetHandles{
        PCI_INTEGER         iReturnCode;
        PCI_INTEGER         MaxHandles;
        BYTE                PCIHandles[COMMON_MAX_SIZE]; //Deep enough to get MaxHandles.
        PCI_INTEGER         ActualHandles;
};

struct IoPciGetProperty {
        PCI_INTEGER         iReturnCode;
        PCI_HANDLE          PCIHandle;
        PCI_INTEGER         MaximumSize;
        BYTE                Property[COMMON_MAX_SIZE]; // Deep enough to get MaxProperty.
        PCI_INTEGER         ActualSize;
};

struct IoPciRegister    {
        PCI_INTEGER         iReturnCode;
        struct pci_register PciRegisterInfo;
        PCI_EXID            ExID;
};
```

```
struct IoPciDeregister{
        PCI_INTEGER     iReturnCode;
        PCI_EXID        ExID;
};

struct IoPciPutMessage{
        PCI_INTEGER     iReturnCode;
        PCI_EXID        ExID;
        PCI_MPB         PCIMPB;
        BYTE            Message[COMMON_MAX_SIZE];
        BYTE            Data[COMMON_MAX_SIZE];
};

struct IoPciGetMessage{
        PCI_INTEGER     iReturnCode;
        PCI_EXID        ExID;
        PCI_MPB         PCIMPB;
        BYTE            Message[COMMON_MAX_SIZE];
        BYTE            Data[COMMON_MAX_SIZE];
};

struct IoPciSetSignal{
        PCI_INTEGER     iReturnCode;
        PCI_EXID        ExID;
        PCI_INTEGER     Signal;
        PCI_PROCEDURE   SignalProcedure;
};
```

7.6.1.2.1 PCI device driver call specification

This paragraph describes IoControl code and buffer format send by user or kernel PUF and received by kernel NAF.

7.6.1.2.1.1 DeviceIoControl parameters

The dwIoControlCode is generated with CTL_CODE macro.

CTL_CODE parameters:

DeviceType 40000.

Method METHOD_BUFFERED (Intermediate kernel system buffer is used).

Access FILE_ANY_ACCESS

Function

PciGetHandles	0x800 (PUF only).	IOCTL_PCIGETHANDLES
PciGetProperty	0x801	IOCTL_PCIGETPROPERTY
PciRegister	0x802	IOCTL_PCIREGISTER
PciDeregister	0x803	IOCTL_PCIDEREGISTER
PciPutMessage	0x804	IOCTL_PCIPUTMESSAGE
PciGetMessage	0x805	IOCTL_PCIGETMESSAGE
PciSetSignal	0x806	IOCTL_PCISETSIGNAL

```
#define IOCTL_PCIREGISTER CTL_CODE(40000, 0x802, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

7.6.1.2.1.2 PCI parameters mapping

As multiple parameters are not available with the DeviceIoControl function, we use the Input buffer to pass PCI USER calls parameters to the NAF device driver part. PCI Parameter shall be copied in the input buffer. In order to simplify calls and buffer mapping, a private Iocal structure is used (see in subclause 7.6.1.2 the device driver structures).

The following sample illustrate this mechanism for the *PciRegister* function.

User mode parameters: struct pci_register *PCIRegisterInfo, PCI_EXID *ExID

The PCI PUFs functions shall copy all parameters content in the DeviceIoControl Input buffer in the function definition order. After the NAF return, the DeviceIoControl Output buffer contains the call result, and the PUF copies Output buffer content in parameters.

DeviceIoControl Input buffer:

SHORT	PCI_INTEGER	**NAF return value**
SHORT	PCI_INTEGER	PUFVersion **pci_register**
SHORT	PCI_INTEGER	PUFType
SHORT	PCI_INTEGER	MaxMsgSize
HINSTANCE	HINSTANCE	DLLInstance, **ExID**
SHORT	PCI_INTEGER	exchange_Id

```
// Sample code
PCI_INTEGER PASCAL PciRegister(struct pci_register *PCIRegisterInfo, PCI_EXID *ExID)
{
PCI_INTEGER                     ReturnCode;
HANDLE              hDD;
struct IoPciRegister    Pr;

        // Open device driver session.
        hDD = CreateFile("",  GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,0,NULL);

        if(hDD != INVALID_HANDLE_VALUE)
                {
                ExID->DLLInstance = (HINSTANCE)hDD;
                // Copy parameters to the input buffer.
                Pr.PciRegisterInfo = *PciRegisterInfo;
                Pr.ExID = *ExID;

                // Call device driver
                DeviceIoControl(        hDD,
                                        IOCTL_PCIREGISTER;
                                        &Pr,
                                        sizeof(struct IoPciRegister),
                                        &Pr,
                                        sizeof(struct IoPciRegister),
                                        &ByteReturned,
                                        NULL);

                // Copy Ouput buffer to return values and ReturnCode.
                *PciRegister  = Pr.PciRegister;
                *ExID = Pr.ExID;
                ReturnCode = Pr.iReturnCode;
                }
return ReturnCode;
}
```

7.6.1.3 Functions description

This subclause describes the implementation, under Windows NT, of the Profile A exchange method functions. During a PUF to NAF call, the stack should have 4K free.

7.6.1.3.1 PciGetHandles

Under Windows NT, The **PciGetHandles** function uses the Registry to locate all information about installed NAFs.

7.6.1.3.1.1 DLL version

Information about available NAFs shall be stored in the following area of the registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PCI\Drivers\pciDriver<number>=DLLName (with number=1..32)
```

As an example, and following Microsoft's convention:

```
\HKEY_LOCAL_MACHINE
   \Software
         \PCI
           \Drivers
               pciDriver1 = 'DLL1.DLL'
               pciDriver2 = 'DLL2.DLL'
```

The following operations shall get all names of installed NAF drivers:

⇒ Open the PCI\Drivers Key of the registry;
⇒ Read each subkey of the previously opened key in the form pciDriver<Number>;
⇒ Close the PCI key of the registry.

7.6.1.3.1.2 Device driver version

Information about available NAFs shall be stored in the following area of the registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PCI\DeviceDrivers\pciDeviceDriver<number>=DLLName (with
number=1..32)
```

As an example, and following Microsoft's convention:

```
\HKEY_LOCAL_MACHINE
   \Software
         \PCI
           \DeviceDrivers
               pciDeviceDriver1 = 'NafName1'
               pciDeviceDriver2 = 'NafName2'
```

The following operations shall get all names of installed NAF drivers:

⇒ open the PCI\DeviceDrivers key of the registry;
⇒ read each subkey of the previously opened key in the form pciDeviceDriver<Number>;
⇒ close the PCI key of the registry;
⇒ test device driver activity by open/close mechanism.

7.6.1.3.2 PciGetProperty

This function is in charge of providing to the PUF the PROPERTY of the NAF. Implicitly, it checks if the NAF is available and loads the library via the **LoadLibrary** function.

7.6.1.3.2.1 DLL version

The following operations shall take place in order:

⇒ load the DLL;
⇒ get the address of the **PciGetProperty** function exported by the NAF;
⇒ call this address with parameters supplied by the PUF;
⇒ finally, free the loaded DLL.

7.6.1.3.2.2 Device driver version

The following operations shall take place in order:

⇒ open a device driver session;
⇒ use **IoPciGetProperty** structure as input and output buffer;
⇒ call the **PciGetProperty** NAF service;
⇒ finally, close the session.

7.6.1.3.3 PciRegister

This function is in charge of providing an association between a PUF and a NAF.

7.6.1.3.3.1 DLL version

The availability of the chosen NAF is checked before loading the library. The NAF is loaded (i.e. the DLL is loaded), and the Handle of the DLL is initialised into the Exchange Identifier structure provided by the caller.

The following operations shall take place in order:

⇒ load the DLL;
⇒ provide the Handle part of the Exchange Identifier with the DLL Handle;
⇒ get the address of the **PciRegister** function exported by the NAF;
⇒ call this address to inform the NAF or a new PUF, supplied parameters are also passed;
⇒ upon return from the NAF, Exchange_Id part of the Exchange Identifier have been provided by the NAF;
⇒ return to the PUF with the returned code from the NAF.

7.6.1.3.3.2 Device driver version

Function open a device driver session, and the Handle of the NAF device driver is initialised into the Exchange Identifier structure provided by the caller.

The following operations shall take place in order:

⇒ open a device driver session;
⇒ use **IoPciRegister** structure as input and output buffer;
⇒ call the **PciRegister** NAF service;
⇒ upon return from the NAF, Exchange_Id part of the Exchange Identifier have been provided by the NAF;
⇒ return to the PUF with the returned code from the NAF.

7.6.1.3.4 PciDeregister

This function is in charge of disassociating a PUF and a NAF. When all the PUF registered have deregistered, the NAF could allow to be unloaded.

7.6.1.3.4.1 DLL version

The following operations should take place, in order:

⇒ get the address of the NAF's PciDeregister;
⇒ call this address to inform the NAF of the end of the association. The address of the PCI_EXID structure is passed to the NAF;
⇒ finally free the DLL if no more PUF is using the DLL.

7.6.1.3.4.2 Device driver version

The following operations should take place, in order:

⇒ get the NAF's device driver handle from ExID structure;
⇒ use **IoPciDeregister** structure as input and output buffer;
⇒ call the **PciDeregister** NAF service;
⇒ NAF service Deregister Exchange_Id part of the Exchange Identifier;
⇒ finally free the device driver session if no more PUF is using the NAF.

7.6.1.3.5 PciPutMessage

This function is in charge of providing a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as indicated in the generic description of the PciPutMessage function.

7.6.1.3.5.1 DLL version

The following operations shall take place, in order:

⇒ get the address of the NAF's PciPutMessage function exported by the NAF;
⇒ call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.6.1.3.5.2 Device driver version

The following operations shall take place, in order:

⇒ get the NAF's device driver handle from ExID structure;
⇒ use **IoPciPutMessge** structure as input and output buffer;
⇒ call the **PciPutMessage** NAF service.

7.6.1.3.6 PciGetMessage

This function is in charge of providing a message, and associated data if any, from a PUF to a NAF. Parameters are provided in the same order as indicated in the generic description of the PciGetMessage function. Buffers provided by the PUF are directly used by the NAF.

7.6.1.3.6.1 DLL version

The following operations shall take place, in order:

⇒ get the address of the NAF's PciGetMessage function exported by the NAF;
⇒ call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.6.1.3.6.2 Device driver version

The following operations shall take place, in order:

⇒ get the NAF's device driver handle from ExID structure;
⇒ use **IoPciGetMessge** structure as input and output buffer;
⇒ call the **PciGetMessage** NAF service.

7.6.1.3.7 PciSetSignal

This function allows a PUF to provide a direct information mechanism to be used by the NAF in case of incoming event.

7.6.1.3.7.1 DLL version

Two mutually exclusive mechanisms are offered under Windows NT:

– a local semaphore mechanism;
– a callback function mechanism.

Once a mechanism is chosen by the PUF, the other is de-activated by the NAF for that particular PUF. Both methods shall be supported by the NAF.

The first mechanism does not use the SignalProcedure parameter. This parameter shall be set to 0. The second mechanism used the SignalProcedure to identify the function to be called by the NAF. In that case, the SignalProcedure parameter shall not be equal to 0.

The following operations shall take place, in order:

⇒ get the address of the NAF's PciSetSignal function exported by the NAF;
⇒ call this address to pass parameter to the NAF (including the address of the PCI_EXID).

7.6.1.3.7.2 Device driver version

A unique event mechanism is specified here.

The following operations shall take place, in order:
⇒ Get the NAF's device driver handle from ExID structure.
⇒ Use **IoPciSetSignal** structure as input and output buffer.
⇒ Call the **PciGetMessage** NAF service.

7.6.1.3.7.3 Signal mechanism

7.6.1.3.7.3.1 DLL version

The semaphore mechanism uses a Handle created by the **CreateSemaphore** function of the Windows NT API. The PUF process can then wait on the semaphore. When an inbound message is available, the NAF will signal the semaphore causing the PUF to wake up and retrieve the message, by calling the **PciGetMessage** function.

The PUF is responsible for giving an initialised semaphore to the NAF in the Signal parameter when calling the function PciSetSignal. After de-registering or de-activating the signal mechanism, the PUF shall also close the local semaphore.

7.6.1.3.7.3.2 Device driver version

The event mechanism uses an event created by the **CreateEvent** function of the Windows NT API. The PUF process can then wait on the event by using the **WaitForSingleObject** function. On signal the PUF wake up and retrieve a message, by calling the **PciGetMessage** function.

The PUF is responsible for giving an initialised event to the NAF in the Signal parameter when calling the function PciSetSignal. After de-registering or de-activating the signal mechanism, the PUF shall also close the local event.

7.6.1.3.7.4 Callback function mechanism

7.6.1.3.7.4.1 DLL version

The PUF supplies a pointer to a PUF resident notification function in the SignalProcedure parameter and a PUF defined context value in the Signal parameter. Each time an inbound message is available, the NAF calls the PUF's notification function with the PUF context value in parameter. The PUF shall then call the **PciGetMessage** function to retrieve the available message.

The call-back function may be called from either the process or interrupt context.

7.6.1.3.7.4.2 Device driver version

No call-back mechanism is available from Kernel to User mode.

7.6.1.3.7.5 De-activation mechanism

7.6.1.3.7.5.1 DLL version

To de-activate any signal mechanism the PciSetSignal function Signal and SignalProcedure parameters shall be NULL. Once deactivated, the previous mechanism shall no longer be used by the NAF to call the PUF.

7.6.1.3.7.5.2 Device driver version

The mechanism is the same as the one used in the DLL version.

7.6.1.4 Availability of NAF's PCI_HANDLE

Under Windows NT, a NAF is loaded only once and the code of the DLL shall be declared as SHARED. The DATA part of the DLL may be declared either as SHARED or not depending on the design of the implementation.

No particular action is required for the NAF when loading or unloading with respect to registry information concerning the NAF. When installing the NAF into the Windows NT system, the registry PCI variables should be initialised as described in this ETS. If the NAF needs to be removed from system, corresponding variables of the registry should be deleted at the same time.

7.6.2 Windows NT for Profile B

7.6.2.1 Windows NT (application level)

Under the operating system Windows NT Version 3.x the Profile B services are provided via a DLL. The interface between applications and Profile B is realised as a function interface. An application can issue Profile B function calls to perform Profile B operations.

The DLL providing the function interface shall be named "CAPI2032.DLL". It is a 32 bit DLL exporting 32 bit APIENTRY type functions.

The DLL functions are exported under following names and ordinal numbers:

CAPI_MANUFACTURER (reserved)	CAPI2032.99
CAPI_REGISTER	CAPI2032.1
CAPI_RELEASE	CAPI2032.2
CAPI_PUT_MESSAGE	CAPI2032.3
CAPI_GET_MESSAGE	CAPI2032.4
CAPI_WAIT_FOR_SIGNAL	CAPI2032.5
CAPI_GET_MANUFACTURER	CAPI2032.6
CAPI_GET_VERSION	CAPI2032.7
CAPI_GET_SERIAL_NUMBER	CAPI2032.8
CAPI_GET_PROFILE	CAPI2032.9
CAPI_INSTALLED	CAPI2032.10

These functions can be called by an application according to the DLL conventions as imported functions.

In the Windows NT environment following types are used to define the functional interface:

WORD	16 bit unsigned integer;
DWORD	32 bit unsigned integer;
PVOID	pointer to any memory location;
PVOID *	pointer to a PVOID;
char*	pointer to a character string;
DWORD *	pointer to a 32 bit unsigned integer value.

7.6.2.1.1 Message operations

7.6.2.1.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the four parameters messageBufferSize, maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs.

For a "normal" application, the size of the message buffer should be calculated using the following formula:

MessageBufferSize = 1 024 + (1 024 * maxLogicalConnection)

Function call

```
DWORD APIENTRY CAPI_REGISTER (      DWORD messageBufferSize,
                                    DWORD maxLogicalConnection,
                                    DWORD maxBDataBlocks,
                                    DWORD maxBDataLen,
                                    DWORD * pApplID);
```

Parameter	Comment
messageBufferSize	Size of Message Buffer
maxLogicalConnection	Maximum number of logical connections
maxBDataBlocks	Number of data blocks available simultaneously
maxBDataLen	Maximum size of a data block
pApplID	Pointer to the location where Profile B should place the assigned application identification number

Return Value

Return Value	Comment
0x0000	Registration successful - application identification number has been assigned
All other values	Coded as described in parameter info class 0x10xx

7.6.2.1.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. Profile B shall release all resources that have been allocated.

The application is identified by the application identification number that had been assigned in the previous CAPI_REGISTER operation.

Function call

```
DWORD APIENTRY CAPI_RELEASE (DWORD ApplID);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER

Return Value

Return Value	Comment
0x0000	Release of the application successful
All other values	Coded as described in parameter info class 0x11xx

7.6.2.1.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an application identification number.

Function call

DWORD APIENTRY CAPI_PUT_MESSAGE (	**DWORD ApplID,** **PVOID pCAPIMessage);**

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
pCAPIMessage	pointer to the message that is passed to Profile B

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

NOTE: When the process returns from the function call the message memory area can be reused by the application.

7.6.2.1.1.4 CAPI_GET_MESSAGE

Description
With this operation the application retrieves a message from Profile B. The application can only retrieve those messages intended for the stipulated application identification number. If there is no message waiting for retrieval, the function returns immediately with an error code.

Function call

DWORD APIENTRY CAPI_GET_MESSAGE (	**DWORD ApplID,** **PVOID * ppCAPIMessage);**

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER
ppCAPIMessage	Pointer to the memory location where Profile B should place the pointer to the retrieved message

Return Value

Return Value	Comment
0x0000	Successful - Message was retrieved from Profile B
All other values	Coded as described in parameter info class 0x11xx

NOTE: The received message may become invalid the next time the application issues a **CAPI_GET_MESSAGE** operation for the same application identification number. This especially matters in multi threaded applications where more than one thread may execute **CAPI_GET_MESSAGE** operations. The synchronisation between threads shall be done by the application.

7.6.2.1.2 Other functions

7.6.2.1.2.1 CAPI_WAIT_FOR_SIGNAL

Description
This operation is used by the application to wait for an asynchronous event from the CAPI.

Function call
This function returns as soon as a message from the CAPI is available.

```
DWORD APIENTRY CAPI_WAIT_FOR_SIGNAL (          DWORD ApplID);
```

Parameter	Comment
ApplID	Application identification number that had been assigned by call of the function CAPI_REGISTER

Return Value

Return Value	Comment
0x0000	No error
All other values	Coded as described in parameter info class 0x11xx

7.6.2.1.2.2 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B (DLL). SzBuffer on call is a pointer to a buffer of 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

Function call

```
VOID APIENTRY CAPI_GET_MANUFACTURER (char* SzBuffer);
```

Parameter	Comment
SzBuffer	Pointer to a buffer of 64 bytes

7.6.2.1.2.3 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

Function call

```
DWORD APIENTRY CAPI_GET_VERSION (      DWORD * pCAPIMajor,
                                       DWORD * pCAPIMinor,
                                       DWORD * pManufacturerMajor,
                                       DWORD * pManufacturerMinor);
```

Parameter	Comment
pCAPIMajor	Pointer to a dword receiving Profile B major version number: 2
pCAPIMinor	Pointer to a dword receiving Profile B minor version number: 0
pManufacturerMajor	Pointer to a dword receiving manufacturer specific major number
pManufacturerMinor	Pointer to a dword receiving manufacturer specific minor number

Return Value

Return	Comment
0x0000	No error, version numbers are copied.

7.6.2.1.2.4 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. SzBuffer on call is a pointer to a buffer of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

Function call

```
DWORD APIENTRY CAPI_GET_SERIAL_NUMBER (char * SzBuffer);
```

Parameter	Comment
SzBuffer	Pointer to a buffer of 8 bytes

Return Value

Return	Comment
0x0000	No error SzBuffer contains the serial number in plain text in the form of a 7-digit number. If no serial number is provided by the manufacturer, an empty string is returned.

7.6.2.1.2.5 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. SzBuffer on call is a pointer to a buffer of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. *CtrlNr* contains the controller number (bit 0..6), for which this information is requested.

```
DWORD APIENTRY CAPI_GET_PROFILE (     PVOID SzBuffer,
                                      DWORD CtrlNr);
```

Parameter	Comment
SzBuffer	Pointer to a buffer of 64 bytes
CtrlNr	Number of Controller. If 0, only number of installed controllers is given to the application.

Return Value

Return	Comment
0x0000	No error
<> 0	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
WORD	number of installed controllers, least significant octet first
WORD	number of supported B-channels, least significant octet first
DWORD	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4..31: reserved
DWORD	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
DWORD	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
DWORD	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
6 DWORDs	reserved for Profile B usage
5 DWORDs	manufacturer specific information
NOTE: This function can be extended, so an application has to ignore unknown bits. Profile B shall set every reserved field to 0.	

7.6.2.1.2.6 CAPI_INSTALLED

Description
This function can be ued by an application to determine if the ISDN hardware and necessary drivers are installed.

Function call

DWORD APIENTRY CAPI_INSTALLED (VOID)

Return Value

Return	Comment
0x0000	Profile B is installed
Any other value	Coded as described in parameter info class 0x11xx

7.6.2.2 Windows NT (device driver level)

For kernel-mode applications, the Profile B for Windows NT shall be implemented as kernel mode device driver. The interface to such a kernel-mode device driver under Windows NT is based on I/O Request Packets (IRPs) which can be sent to the driver, by either kernel-mode or user mode applications.

A CAPI20 device driver creates at least one CAPI20 device object which can be addressed by an application. The ***Flags*** field of each device object shall be ORed with DO_DIRECT_IO after creation. For identification each device object is given a name. The name of a CAPI20 device object is \Device\CAPI20x, where x is a configured decimal ordinal number. The CAPI20 device object name can be used by kernel-mode applications to send IRPs to the corresponding CAPI20 device driver.

A CAPI20 device driver can support multiple controllers. The implementation is free to create a single device object for all supported controllers or a separate device object for each supported controller. The controller numbers are assigned per CAPI20 device object starting with 1.

In order to be accessible by user-mode applications, a CAPI20 device driver creates a symbolic link object for each CAPI20 device object. The name of the symbolic link object is \DosDevices\CAPI20x where x is the same ordinal number as used for the device object name. This allows user-mode applications to get access to the driver provided Profile B services using the name \\.\CAPI20x in a Win32 *CreateFile()* operation.

To ensure the correct loading order of a CAPI20 driver, the driver shall be assigned to the group ´CAPI20´. This is achieved by adding the REG_SZ value entry ‘Group’ to the drivers service subkey in the register.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\<CAPI-Driver-Service>\

DisplayName:REG_SZ: CAPI20 Driver ...
ErrorControl:REG_DWORD: ...
Group: REG_SZ: CAPI20
ImagePath:REG_SZ: ...
Start:REG_DWORD: ...
Type:REG_DWORD: ...

The driver installation shall ensure that the group CAPI20 is listed in ServiceGroupOrder immediately before the Group NDIS.

To permit the unambigous configuration of all CAPI20 device drivers a new comon subkey is created in the Windows NT registry. This subkey is named CAPI20 and contains a subkey x for each CAPI20x device object created by CAPI20 device drivers. During installation of a new CAPI20 device driver the CAPI20 subkey shall be queried. If the CAPI20 subkey not exists already it shall be created by the installation procedure. For each device object created by the new driver a new subkey x is created with the lowest possible ordinal number: The ordinal number for the first CAPI20 device object is 1.Thus the first installed CAPI20 device driver shall use the name \Device\CAPI201 for its first device object and the name\Device\CAPI202 for its the second device object (if any) etc. The ordinal numbers claimed by the new driver shall be remembered in the drivers private configuration data. When the driver is removed from the system the deinstallation procedure shall remove the corresponding subkeys under the CAPI20 subkey too.

HKEY_LOCAL_MACHINE\SOFTWARE\CAPI20\

Contents:
1\
NumberOfControllers: REG_DWORD: <Number of supported Controllers>
Manufacturer: REG_SZ: <Manufacturer Name>
DeviceName: REG_SZ: CAPI201

/* For each supported controller a controller subkey is creadted: */
1\
Channels: REG_DWORD: <Number of B-channels supported by this controller>
2\
etc.
2\ ...

Every CAPI20 conformant driver shall be prepared to work in a chain of layered drivers. Thus the driver shall not use any operation which is legal only for a highest level driver.

Every CAPI20 conformant driver shall be prepared to be unloaded, i.e. the driver shall set the entry point of his Unload routine in the DriverObject passed to his DriverEntry routine. The Unload routine shall release all previously claimed kernel and hardware resources to permit an new initialization of the driver at a later time.

Every CAPI20 conformant driver shall handle the cancellation of Irp´s.

Every CAPI20 conformant driver shall handle the following major function codes:

IRP_MJ_CREATE
IRP_MJ_CLEANUP
IRP_MJ_CLOSE
IRP_MJ_READ
IRP_MJ_WRITE
IRP_MJ_SHUTDOWN
IRP_MJ_DEVICE_CONTROL
IRP_MJ_INTERNAL_DEVICE_CONTROL

To receive shutdown notifications from the system shutdown processing in a highest level driver, the driver shall call IoShutdownNotification() in its DriverEntry routine but shall ignore any error returned by this call.

Three types of IRP_MJ_xxx functions are used by an user mode application to communicate with the Profile B device: IRP_MJ_DEVICE_CONTROL, IRP_MJ_READ and IRP_MJ_WRITE.

The IRP_MJ_INTERNAL_DEVICE_CONTROL function is reserved for the exclusive use by kernel mode applications, i.e for inter device driver communication.

IRP_MJ_DEVICE_CONTROL is used for all CAPI20 functions except CAPI_GET_MESSAGE and CAPI_PUT_MESSAGE.

The CAPI_GET_MESSAGE and CAPI_PUT_MESSAGE functions use IRP_MJ_READ/WRITE (user-mode and kernel-mode applications) or IRP_MJ_INTERNAL_DEVICE_CONTROL (kernel mode applications only).

The following DEVICE_CONTROL and INTERNAL_DEVICE_CONTROL codes are defined for the Profile B functions:

```
/*
*       the common device type code for CAPI20 conformant drivers
*/
#define FILE_DEVICE_CAPI20 0x8001

/*
```

```
*           DEVICE_CONTROL codes for user AND kernel mode applications
*/
#define CAPI20_CTL_BASE                 0x800
#define CAPI20_CTL_REGISTER             (CAPI20_CTL_BASE+0x0001)
#define CAPI20_CTL_RELEASE              (CAPI20_CTL_BASE+0x0002)
#define CAPI20_CTL_GET_MANUFACTURER     (CAPI20_CTL_BASE+0x0005)
#define CAPI20_CTL_GET_VERSION          (CAPI20_CTL_BASE+0x0006)
#define CAPI20_CTL_GET_SERIAL           (CAPI20_CTL_BASE+0x0007)
#define CAPI20_CTL_GET_PROFILE          (CAPI20_CTL_BASE+0x0008)

/*
*           INTERNAL_DEVICE_CONTROL codes for kernel mode applications only
*/
#define CAPI20_CTL_PUT_MESSAGE          (CAPI20_CTL_BASE+0x0003)
#define CAPI20_CTL_GET_MESSAGE          (CAPI20_CTL_BASE+0x0004)

/*
*           The wrapped control codes as required by the system
*/
#define CAPI20_CTL_CODE(function,method) \
            CTL_CODE(FILE_DEVICE_CAPI20,function,method,FILE_ANY_ACCESS)

#define IOCTL_CAPI_REGISTER \
        CAPI20_CTL_CODE(CAPI20_CTL_REGISTER, METHOD_BUFFERED)

#define IOCTL_CAPI_RELEASE \
        CAPI20_CTL_CODE(CAPI20_CTL_RELEASE, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_MANUFACTURER\
        CAPI20_CTL_CODE(CAPI20_CTL_GET_MANUFACTURER, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_VERSION \
        CAPI20_CTL_CODE(CAPI20_CTL_GET_VERSION, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_SERIAL \
        CAPI20_CTL_CODE(CAPI20_CTL_GET_SERIAL, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_PROFILE \
        CAPI20_CTL_CODE(CAPI20_CTL_GET_PROFILE, METHOD_BUFFERED)

#define IOCTL_CAPI_MANUFACTURER \
        CAPI20_CTL_CODE(CAPI20_CTL_MANUFACTURER, METHOD_BUFFERED)

#define IOCTL_CAPI_PUT_MESSAGE \
        CAPI20_CTL_CODE(CAPI20_CTL_PUT_MESSAGE, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_MESSAGE \
        CAPI20_CTL_CODE(CAPI20_CTL_GET_MESSAGE, METHOD_BUFFERED)
```

To transfer CAPI20 specific return values from the driver to kernel- or user-mode applications the status code of the IRP is set accordingly. Cause only some IRP status codes are mapped directly to Win32 error codes (return codes of DeviceIoControl(), ReadFile(), WriteFile()) the following status code representation for CAPI20 errors (Info values) shall be used:

Info	Windows NT Status code	Win32 Error Code
0x1001	STATUS_TOO_MANY_SESSIONS	ERROR_TOO_MANY_SESSIONS
0x1002	STATUS_INVALID_PARAMETER	ERROR_INVALID_PARAMETER
0x1003	N.A.	
0x1004	STATUS_BUFFER_TOO_SMALL	ERROR_INSUFFICIENT_BUFFER
0x1005	STATUS_NOT_SUPPORTED	ERROR_NOT_SUPPORTED
0x1006	N.A.	
0x1007	STATUS_NETWORK_BUSY	ERROR_NETWORK_BUSY
0x1008	STATUS_INSUFFICIENT_RESOURCES	ERROR_NOT_ENOUGH_MEMORY
0x1009	N.A.	
0x100a	STATUS_SERVER_DISABLED	ERROR_SERVER_DISABLED
0x100b	STATUS_SERVER_NOT_DISABLED	ERROR_SERVER_NOT_DISABLED
0x1101	STATUS_INVALID_HANDLE	ERROR_INVALID_HANDLE
0x1102	STAUS_ILLEGAL_FUNCTION	ERROR_INVALID_FUNCTION
0x1103	STATUS_TOO_MANY_COMMANDS	ERROR_TOO_MANY_CMDS
0x1104	N.A.	
0x1105	STATUS_DATA_OVERRUN	ERROR_IO_DEVICE
0x1106	STATUS_INVALID_PARAMETER	STATUS_INVALID_PARAMETER
0x1107	STATUS_DEVICE_BUSY	ERROR_BUSY
0x1108	STATUS_INSUFFICIENT_RESOURCES	ERROR_NOT_ENOUGH_MEMORY
0x1109	N.A	
0x110a	STATUS_SERVER_DISABLED	ERROR_SERVER_DISABLED
0x100b	STATUS_SERVER_NOT_DISABLED	ERROR_SERVER_NOT_DISABLED

In Windows NT all the communication between a device object and an application is related to a file object. For that reason a file object pointer (or a file handle) is used to identify a link between a CAPI20 device object and an user-mode or kernel-mode application instead of the application id. Any application ids within Profile B messages are therefore ignored.

In the following the interface between the application and the Profile B device driver is described with Win32 functions. These functions are available for user-mode applications only. The equivalent kernel mode functions can be found in the Windows NT documentation.

7.6.2.2.1 Message operations

7.6.2.2.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the four parameters MessageBufferSize,maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs for the connections it is going to accept or it will try to establish itself.

CAPI_REGISTER	CAPI_CTL_REGISTER

Implementation
For the CAPI_REGISTER function the application shall get a handle to the Profile B device using the Win32 CreateFile function and then send a CAPI_CTL_REGISTER to the Profile B device. With CAPI_REGISTER the following data structure is passed on to the driver:

```
struct capi_register_params {
        WORD MessageBufferSize,
        WORD maxLogicalConnection,
        WORD maxBDataBlocks,
        WORD maxBDataLen
};
```

Only one CAPI_CTL_REGISTER may be sent with one handle if no CAPI_CTL_RELEASE is sent in between. If an application program wants to register as more than one Profile B application, it shall obtain several handles with CreateFile and send one CAPI_CTL_REGISTER with each handle.

EXAMPLE:

```
capi_handle = CreateFile("\\\\.\\CAPI201",
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
                NULL);
r.MessageBufferSize = MessageBufferSize;
r.maxLogicalConnections = maxLogicalConnections;
r.maxBDataBlock = maxBDataBlocks;
r.maxBDataLen = maxBDataLen;
ret = DeviceIoControl(capi_handle,
            CAPI_CTL_REGISTER,
            &r,
            sizeof(struct capi_register_params),
            NULL,
            0,
            &ret_bytes,
            NULL);
```

7.6.2.2.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. This way the application signals Profile B that all resources that have been allocated by Profile B for the application can be released again.

CAPI_RELEASE	**CAPI_CTL_RELEASE**

Implementation
A CAPI_RELEASE can be performed in two ways. If the same handle is to be used again, an CAPI_CTL_RELEASE shall be sent. If the handle is not used any more the handle may just be closed using CloseHandle.

EXAMPLE:

```
ret = DeviceIoControl(capi_handle,
                CAPI_CTL_RELEASE,
                NULL,
                0,
                NULL,
                0,
                &ret_bytes,
                NULL);
or

CloseHandle(capi_handle);
```

7.6.2.2.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an file handle.

CAPI_PUT_MESSAGE	**WriteFile()/CAPI_CTL_PUT_MESSAGE**

Implementation
The CAPI_PUT_MESSAGE function can be performed by using either a WriteFile() operation or a INTERNAL_DEVICE_CONTROL IRP. The INTERNAL_DEVICE_CONTROL method is available to kernel mode applications only.

1) WriteFile() operation

With the WriteFile() operation one data buffer is sent to the CAPI20 device driver. This buffer shall contain the message AND data associated with a DATA_B3_REQ message. The data (if available) shall be placed into the buffer immediately following the message.

```
ret = WriteFile(capi_handle,
                (PVOID)msg,     /* buffer for message + data */
                msg_length,     /* length of message + data */
                &ret_bytes,
                                &o_write);
```

The WriteFile() operation completes immediately, it does not wait for any network event. (normal CAPI_PUT operation).

The buffer can be re-used by the application, as soon as the WriteFile() operation completes.

2) INTERNAL_DEVICE_CONTROL

Kernel mode applications may use an INTERNAL_DEVICE_CONTROL IRP with an IO_CONTROL code CAPI_CTL_PUT_MESSAGE for the CAPI_PUT_MESSAGE Operation. With this IRP a pointer to the following structure is passed in Parameters.DeviceControl.Type3InputBuffer to the CAPI20 device driver:

```
struct {
        PVOID message;
        PVOID data;
};
```

The buffer passed in the ´message´ field can be re-used by the application, as soon as the INTERNAL_DEVICE_CONTROL completes. The buffer passed in the ´data´ field can be re-used by the application as soon as the corresponding DATA_B3_CONF message is received.

7.6.2.2.1.4 CAPI_GET_MESSAGE

Description
With this operation the application retrieves a message from Profile B. The application retrieves all messages associated with the corresponding file handle from operation **CAPI_REGISTER**.

CAPI_GET_MESSAGE	**ReadFile()/CAPI_CTL_GET_MESSAGE**

Implementation
The CAPI_GET_MESSAGE function can be performed by using either a ReadFile() operation or a INTERNAL_DEVICE_CONTROL IRP. The INTERNAL_DEVICE_CONTROL method is available to kernel mode applications only.

1) ReadFile() operation

With the ReadFile() operation one data buffer is received from the CAPI20 device driver. This buffer contains the message AND data associated with a DATA_B3_IND message. The data (if available) is placed into the buffer immediately following the message.

```
ret = ReadFile(capi_handle,
        buffer,
        buffer_size,
        &ret_bytes,
        &o_read);
```

The ReadFile() operation completes as soon as a CAPI message is available.

The size of the buffer provided by the application should be at least MessageBufferSize+512. If the buffer provided by the application is too small to hold the message and the data, an error shall be returned and the excess data is lost.

2) INTERNAL_DEVICE_CONTROL

Kernel mode applications may use an INTERNAL_DEVICE_CONTROL IRP with an IO_CONTROL code CAPI_CTL_GET_MESSAGE for the CAPI_GET_MESSAGE Operation. With this IRP a pointer to the following structure is passed in Parameters.DeviceControl.Type3InputBuffer to the CAPI20 device:

```
struct {
        PVOID message;
        PVOID data;
};
```

The CAPI20 device driver fills in the fields of this structure. When the INTERNAL_DEVICE_CONTROL completes the field message contains a pointer to the CAPI message and the field data contains a pointer to a data buffer associated with a DATA_B3_IND.

The message buffer can be re-used by the CAPI20 driver as soon as the application sends the next CAPI_CTL_GET_MESSAGE.

The data buffer can be re-used byte the CAPI20 driver as soon as the applications sends a corresponding DATA_B3_RES message.

7.6.2.2.1.5 CAPI_SET_SIGNAL

There is no CAPI_SET_SIGNAL function. The asynchronous signalling of a received message is done implicitly by completing the corresponding operation (ReadFile() or INTERNAL_DEVICE_CONTROL).

7.6.2.2.2 Other functions

7.6.2.2.2.1 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B. The offered buffer shall have a size of at least 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

CAPI_GET_MANUFACTURER	CAPI_CTL_GET_MANUFACTURER

Implementation
With this IO_CONTROL the manufacturer identification is read from the Profile B driver. A buffer of 64 bytes shall be provided by the application. The manufacturer identification is returned as zero terminated ASCII string.

7.6.2.2.2.2 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

CAPI_GET_VERSION	CAPI_CTL_GET_VERSION

Implementation
With this IO_CONTROL the version of the Profile B can be read. A buffer with the following structure shall be provided by the application:

```
struct capi_version_params {
        word CAPIMajor;
        word CAPIMinor;
        word ManufacturerMajor;
        word ManufacturerMinor;
};
```

7.6.2.2.2.3 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. The offered buffer shall have a size of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

CAPI_GET_SERIAL_NUMBER	CAPI_CTL_GET_SERIAL_NUMBER

Implementation
With this IO_CONTROL the Profile B serial number can be read from the driver. A buffer of 8 bytes shall be provided by the application. The serial number is returned as a zero terminated ASCII string.

7.6.2.2.2.4 CAPI_GET_PROFILE

Description
With this IO_CONTROL the Profile B capabilities can be read from the driver. A buffer of 64 bytes shall be provided by the application. The same profile structure as for the Windows NT application level interface is returned by the driver.

CAPI_GET_PROFILE	CAPI_CTL_GET_PROFILE

Implementation
With this IO_CONTROL the Profile B capabilities can be read from the driver.

7.7 Windows 95

7.7.1 Windows 95 specific implementation for Profile A

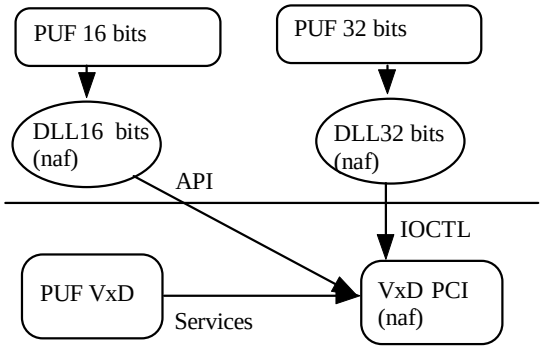
7.7.1.1 Windows 95 Operating System specific implementation for Profile A

7.7.1.1.1 Introduction

Under Windows 95, every NAF shall provide three basic mechanisms to support the ISDN PCI exchange method:

- a PCI DLL (16 bits) to allow user applications (16 bits) access to the ISDN PCI features;
- a PCI DLL (32 bits) to allow user applications (32 bits) access to the ISDN PCI features;
- a PCI VxD to provide ISDN PCI features to other virtual devices and the PCI DLL described below. This VxD offers virtual device services to other virtual device, a virtual device API to the 16 bits PCI DLL and I/O Control to the 32 bits PCI DLL.

User mode applications (16 bits or 32 bits) shall use the DLL mechanism (resp 16 or 32 bits) to access ISDN PCI features.



7.7.1.1.2 Description of the PCI DLL (16 bits)

The PCI DLL (16 bits) is designed to offer the DLL mechanism of Profile A, as described in subclause 7.2.1, without modification for the user application. The application interface is identical as under Windows 3.x but the PCI DLL shall map every ISDN PCI function using the virtual device API exported by the VxD. The Windows 95 DLL naming convention is the same as under Windows 3.x.

7.7.1.1.3 Description of the PCI DLL (32 bits)

The PCI DLL 32 bits is designed to offer the DLL mechanism of Profile for 32 bits PUFs. The PCI DLL shall map every ISDN PCI function using the device IOCTL interface.

7.7.1.1.3 Description of the VxD

The VxD offers three different accesses to its features. Each access is designed to provide the most efficient interface depending on the calling PUF processor mode. The three accesses are the following:

- The VxD shall export an API for the 16 bits PCI DLL;
- The VxD shall implement the device IOCTL interface for the 32 bits PCI DLL;
- The VxD shall offer virtual device services for other VxDs.

Each kind of access will be described in a further subclause.

7.7.1.1.3.1 Virtual Device API

The 16 bits PCI DLL shall use protected mode API procedures of the VxD. To make these procedures available, the PCI VxD shall declare them as parameters in the Declare_Virtual_Device macro as follows:

```
Declare_Virtual_Device                  VPCID,          // Name of the VxD
                                VERS_MAJ, \             // Major number version
                                VERS_MIN, \             // Minor number version
                                VPCID_Control, \        // Event handler
                                VPCID_DEVICE_ID,\       // Device Identifier
                                Undefined_Init_Order, \ // Intialisation order
                                VPCID_handler, \        // Real mode handler
                                VPCID_handler, \        // Protected mode handler
```

The PCI DLL retrieves an entry point address for an API procedure for a given virtual machine by calling the Get Device Entry Point Address function (Interrupt 2Fh Function 1 684h). The BX register is set to the identifier for the virtual device. The Virtual Machine Manager (VMM) returns an address to the DLL to indirectly enter the API procedure.

This can easily be wrapped into a GetPCIEntryPoint function in C as follows:

```
typedef unsigned long (far *PCI_ENTRY_POINT)(void);
PCI_ENTRY_POINT GetPCIEntryPoint(WORD wDeviceID)
        {
        struct SREGS sRegister;
        union REGS uRegister;
        memset(&sRegister,0,sizeof(sRegister));
        uRegister.x.ax = 0x1684;                                        // Get device entry point
        uRegister.x.bx = wDeviceID;                                     // VxD ID number
        int86x(0x2F,&uRegister,&uRegister,&sRegister);                  // Generate INT 2F
        return((PCI_ENTRY_POINT)MK_FP(sRegister.es,uRegister.di));      // returned in ES:DI
        }
```

When the DLL calls the entry point address, the AX register contains one of the following PCI function number:

```
#define PCI_GETHANDLES          0x800
#define PCI_GETPROPERTY         0x801
#define PCI_REGISTER            0x802
#define PCI_DEREGISTER          0x803
#define PCI_PUTMESSAGE          0x804
#define PCI_GETMESSAGE          0x805
#define PCI_SETSIGNAL           0x806
```

The DLL shall use the Parameter Input structures defined in subclause 7.7.1.1.3.2 Device IOCTL interface and place the address of this structure in ES:BX as follows:

```
BOOL  VxDAPICall(void far * fp)
        {
        _asm les bx, dword ptr fp
        _asm mov ax, VPCID_DEVICE_ID
        _asm call dword ptr [API] // API =  GetPCIEntryPoint
        _asm jc Error
        return TRUE;
        Error:
        return FALSE;
        }
```

The VMM saves the registers and calls the PCI VxD corresponding API procedure, placing the handle of the current virtual machine in the EBX register and the address of a Client_Reg_Struc structure in the EBP register. This structure is defined in vmm.h. The PCI VxD shall set its return values in the Client_Reg_Struc structure.

7.7.1.1.3.2 Device IOCTL interface

The virtual device API access is not available for the 32 bits PCI DLL, the VMM assumes it was called by a 16 bits application and crashes in this case. The 32 bits PCI DLL shall use device I/O control to access the PCI VxD. Windows 95 implements the interface through the DeviceIoControl function which sends commands and data directly to the VxD. When the DeviceIoControl function fails, the GetLastError function retrieves the error value. The CreateFile function is the way to obtain a handle on the PCI VxD. Once the device handle is available, it can be used in a call to DeviceIoControl:

```
BOOL DeviceIoControl (HANDLE hDevice,
        DWORD PciCode,
        LPVOID InputParameter,          // Structure which depends on the called exchange method function and contains
                                        // the input parameters
        DWORD InputParameterSize,       // Size of the InputParameter structure
        LPVOID OutputParameter,         // Structure which depends on the called exchange method function and
contains
                                        // the output parameters
        DWORD OutputParameterSize,      // Size of the OutputParameter structure
        LPDWORD pnOut,                  //  Number of bytes actually placed into the OutputParameter structure
        LPOVERLAPPED lpovlap            // set only if asynchronous mode (see 7.7.1.4.7.1.2)
        );
```

In this call, the parameter PciCode is a code the PCI VxD uses to identify the exchange method function to process.

The 32 bits PCI DLL's call to DeviceIoControl turns into a W32-DeviceIoControl event message to the PCI VxD. The VxD receives a DIOC structure in ESI with all the parameters described before.

The PciCode are the following:

```
#define PCI_CTL_GETHANDLES   0x800
#define PCI_CTL_GETPROPERTY 0x801
#define PCI_CTL_REGISTER     0x802
#define PCI_CTL_DEREGISTER   0x803
#define PCI_CTL_PUTMESSAGE   0x804
#define PCI_CTL_GETMESSAGE   0x805
#define PCI_CTL_SETSIGNAL    0x806
```

The Parameter Input structures are the following:

```
#define COMMON_MAX_SIZE     4096
struct IoPciRegister
{
        PCI_INTEGER         iReturnCode;
        struct pci_register PCIRegisterInfo;
        PCI_EXID            ExID;
}

struct IoPciDeregister
{
        PCI_INTEGER         iReturnCode;
        PCI_EXID            ExID;
}

struct IoPciGetProperty
{
        PCI_INTEGER         iReturnCode;
        PCI_HANDLE          PCIHandle;
        PCI_INTEGER         MaximumSize;
        BYTE                Property[COMMON_MAX_SIZE];
        PCI_INTEGER         ActualSize;
}

struct IoPciPutMessage
{
        PCI_INTEGER         iReturnCode;
        PCI_EXID            ExID;
        PCI_MPB             PCIMPB;
        BYTE                Message[COMMON_MAX_SIZE];
        BYTE                Data[COMMON_MAX_SIZE];
}

struct IoPciGetMessage
{
        PCI_INTEGER         iReturnCode;
        PCI_EXID            ExID;
        PCI_MPB             PCIMPB;
        BYTE                Message[COMMON_MAX_SIZE];
        BYTE                Data[COMMON_MAX_SIZE];

}
```

PciGetHandles and PciSetSignal are only used by the PUF. See subclause 7.7.1.4.7.1.2 for details on how the DLL deals with the signal mechanism.

7.7.1.1.3.3 Virtual Device Services

Other virtual devices should directly access the PCI VxD by calling the VxDcall macro. The PCI VxD shall use the Begin_Service_Table and End_Service_Table macro to declare its services. The declaration shall be made as follows:

```
VPCID_DEVICE_ID EQU ??h     //// Every PCI VxD shall provide a unique device identifier. If two different NAF provide the
                            //// same device identifier, they will not be able to be used simultanously.
Begin_Service_Table VPCID
        VPCID_Service VPCID_GetVersion,LOCAL
        VPCID_Service VPCID_MessageOperations,LOCAL
End_Service_Table VPCID
```

This service table declaration shall be placed in an include file so that other virtual devices are able to import PCI services by including the file rather than recreating the declaration. If a virtual device includes many services table declarations, it shall change VPCID in VxxxD where xxx represents every NAF provider. There is another method for a virtual device which wants to dynamically choose its NAF VxD. First it shall include the following service table declaration:

```
#define VPCID_DEVICE_ID  UNDEFINED_DEVICE_ID

Begin_Service_Table     (VPCID, VxD)
        Declare_Service (VPCID_GetVersion, LOCAL)
        Declare_Service (VPCID_MessageOperations, LOCAL)
End_Service_Table       (VPCID, VXD)
```

The VxDCall generates the following code, namely an INT 20H instruction that's followed in memory by a 32-bit service identifier. The high-order 16 bits of this identifier uniquely identify the VxD being called. The low-order 16 bits index a transfer vector defined by the called VxD. A PUF VxD shall defines a VxDPCICall macro in order to call a NAF VxD with a device identifier retrieved by a pciGetHandles call. The VxDPCICall macro shall be as follows:

```
#define VxDPCICall(pci_device_id,service) \
        _asm mov ebx, pci_device_id \
        _asm mov [label], ebx \
        _asm _emit 0xcd \
        _asm _emit 0x20 \
        _asm _emit (GetVxDServiceOrdinal(service)& 0xff) \
        _asm _emit (GetVxDServiceOrdinal(service) >> 8) & 0xff \
        _asm label: \
        _asm _emit 0x90  \
        _asm _emit 0x90
```

7.7.1.1.3.3.1 VPCID_GetVersion service

Depending on a computer's configuration, a VxD may fail to be loaded when Windows 95 starts. This means a virtual device that uses services provided by the PCI VxD shall verify that those PCI services are available before calling them. To verify the PCI services, the calling virtual device shall attempt to call the Get_Version Service of the PCI VxD. If the PCI VxD has not been loaded, the VMM (Virtual Machine Manager) sets the carry flag and returns zero in the AX register. If the PCI VxD has been loaded, it clears the carry flag and returns its version number in the AX register.

7.7.1.1.3.3.2 VPCID_MessageOperations service

This service is a register based service. All the parameters are passed in registers and results are returned in registers as described in VxD subclauses of 7.7.1.4. The service shall preserve all registers it does not explicitly use to return values in. The AL register contains the PCI function number and other client registers are used for additional parameters. PCI function return code shall set in the AX register.

7.7.1.2 Implementation of basic type

```
        typedef short       PCI_INTEGER
        typedef char far *  PCI_BYTEARRAY
#ifdef VxD
        typedef DWORD       PCI_HANDLE      // Device Identifier
#else
        typedef char far *  PCI_HANDLE      // name of the NAF dll. Maximum length of the complete DLL name is
                                            // fixed to 256
#endif

typedef struct      {
                    HANDLE DLLInstance;
                    PCI_INTEGER Exchange_Id;
                    } PCI_EXID;

typedef void (far pascal * PCI_PROCEDURE)();

// ----- PCI Structures

struct pci_mpb
        {
        PCI_INTEGER     MessageID;
        PCI_INTEGER     MessageMaximumSize;
        PCI_INTEGER     MessageActualUsedSize;
        PCI_INTEGER     DataMaximumSize;
        PCI_INTEGER     DataActualUsedSize;
        };

typedef struct pci_mpb PCI_MPB;

struct pci_register             // Structure containing registering info.
        {
        PCI_INTEGER     PUFVersion;     // Give PUF version.
        PCI_INTEGER     PUFType;        // Give PUF type.
        PCI_INTEGER     MaxMsgSize;     // return: max. size of a message.
        };

typedef struct pci_register PCI_REGISTER;

struct pci_opsys                        // Structure containing registering info.
        {
        int     DummyParameter;         // No specific requirement for Windows 95.
        };

typedef struct pci_opsys PCI_OPSYS;
```

7.7.1.3 C Function prototypes

```
PCI_INTEGER  PciGetHandles  (
        PCI_INTEGER         MaxHandles,
        PCI_BYTEARRAY       PCIHandles,
        PCI_INTEGER         far * ActualHandles     );

PCI_INTEGER  PciGetProperty (
        PCI_HANDLE          PCIHandle,
        PCI_INTEGER         MaximumSize,
        PCI_BYTEARRAY       Property,
        PCI_INTEGER         *ActualSize             );

PCI_INTEGER  PciRegister    (
        PCI_HANDLE          PCIHandle,
        struct pci_register *PCIRegisterInfo,
        struct pci_opsys    * PCIOpSysInfo,
        PCI_EXID  *ExID                             );

PCI_INTEGER  PciDeregister  (
        PCI_EXID            *ExID                   );

PCI_INTEGER  PciPutMessage (
        PCI_EXID            *ExID,
        PCI_MPB             *PCIMPB,
        PCI_BYTEARRAY       Message,
        PCI_BYTEARRAY       Data                    );

PCI_INTEGER  PciGetMessage (
        PCI_EXID            *ExID,
```

```
            PCI_MPB             *PCIMPB,
            PCI_BYTEARRAY       Message,
            PCI_BYTEARRAY       Data                );

PCI_INTEGER  PciSetSignal       (
            PCI_EXID            *ExID,
            PCI_INTEGER         Signal,
            PCI_PROCEDURE       SignalProcedure     );
```

7.7.1.4 Description of functions

This subclause describes the implementation under Windows 95 of ISDN PCI exchange method functions. For 16 bits PUFs, functions are the same as described in subclause 7.2.1.4. For 32 bits PUFs and VxD PUFs, functions are described in the following subclauses.

7.7.1.4.1 PciGetHandles

This function provides a way to get all available NAFs.

7.7.1.4.1.1 16 bits PUF

The description is identical as in subclause 7.2.1.4.1.

7.7.1.4.1.2 32 bits PUF

Under Win95, the PciGetHandles uses a PCI.INI file in the WINDOWS directory to get available PCI_HANDLEs. The section [Drivers32] in the PCI.INI file contains all entries of installed NAFs. Each entry has the following format:

pciDriver<number> = DLLName(number=1..32)

The following operations shall get all names of installed NAF drivers:

loops from 1 to 32
- constructs the keyName « pciDriver » associated with the current loop value;
- issues a GetPrivateProfileString using:
  sectionKey = « DRIVERS 32»,
  the keyName constructed before,
  no default value,
  a maximum size equal to 256,
  FileName=« PCI.INI ».

7.7.1.4.1.3 VxD

Under Win95, the PciGetHandles checks the registry to see the value under the PciDriverIdentifier<number> key in:

HKLM\System\CurrentControlSet\Control\SessionManager\KnownVxDs.

The following operations shall get all identifiers of installed NAF VxD:

- open the KnownVxDs Key of the registry;
- read each subkey of the previously opened key in the form pciDriver<number>;
- close the PCI key of the registry.

7.7.1.4.2 PciGetProperty

This function provides the PUF with the property of the NAF.

7.7.1.4.2.1 16 bits PUF

The description is identical as in subclause 7.2.1.4.2.

7.7.1.4.2.2 32 bits PUF

The following operations shall take place in order:

a) load the 32 bits PCI DLL;
b) call the pciGetProperty exported by this DLL;
c) free the 32 bits PCI DLL.

7.7.1.4.2.3 VxD

The following operations shall take place in order:

a) call the GetVersion Service with VxDcall macro;
b) verify that the VxD has been loaded;
c) call the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x01	
PCIHandle	ECX	Call value	
MaximumSize	EDX	Call value	
NAFProperty	ESI	Return value	
ActualSize	EDI	Return value	

The function return value (listed below) is set in the AX register

Success
InvalidPCIHandle
NAFnotavailable
NAFBusy
PropertyBufferTooSmall

7.7.1.4.3 PciRegister

This function provides an association between a PUF and a NAF.

7.7.1.4.3.1 16 bits PUF

The description is identical as in subclause 7.2.1.4.3.

7.7.1.4.3.2 32 bits PUF

The following operations shall take place in order:

- load the DLL;
- provide the DLL instance part of the exchange Identifier with the DLL instance;
- call the pciRegister function exported by the 32 bits PCI DLL.

7.7.1.4.3.3 VxD

The PUF calls the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x02	
PCIHandle	ECX	Call value	
PCIRegisterInfo	EDX	Call and return value	
PCIOpSysInfo	ESI	Call value	
ExID	EDI	Return value	

The function return value (listed below) is set in the AX register

Success
InvalidPCIHandle
NAFnotavailable
NAFBusy
MaxPUFsExceeded
InvalidPUFType
InvalidPUFVersion
InvalidRegisterInfoStructure
InvalidOpSysInfoStructure

7.7.1.4.4 PciDeregister

This function breaks an existing association between a PUF and a NAF.

7.7.1.4.4.1 16 bits PUF

The description is identical to that in subclause 7.2.1.4.4.

7.7.1.4.4.2 32 bits PUF

The following operations shall take place in order:

a) call the PciDeregister function exported by the 32 bits PCI DLL;
b) free the 32 bits PCI DLL.

7.7.1.4.4.3 VxD

The PUF calls the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x03	
ExID	EDI	Call value	

The function return value (listed below) is set in the AX register

Success
InvalidExID
NAFnotavailable
NAFBusy

7.7.1.4.5 PciPutMessage

This function provides a message and associated data from a PUF to a NAF.

7.7.1.4.5.1 16 bits PUF

The description is identical as in subclause 7.2.1.4.5.

7.7.1.4.5.2 32 bits PUF

The PUF calls the PciPutMessage function exported by the 32 bits PCI DLL.

7.7.1.4.5.3 VxD

The PUF calls the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x04	
PCIMBP	ECX	Call value	
Message	EDX	Call value	
Data	ESI	Call value	
ExID	EDI	Call value	

The function return value (listed below)is set in the AX register.

Success
InvalidExID
NAFnotavailable
NAFBusy
InvalidPCIMPB
InvalidMessageBuffer
PCIMBPTooSmall
MessageBufferTooSmall
MessageTooLarge
DataBufferRequired

7.7.1.4.6 PciGetMessage

This function provides a message and associated data from a NAF to a PUF.

7.7.1.4.6.1 16 bits PUF

The description is identical as in subclause 7.2.1.4.6.

7.7.1.4.6.2 32 bits PUF

The PUF calls the PciGetMessage function exported by the 32 bits PCI DLL.

7.7.1.4.6.3 VxD

The PUF calls the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x05	
PCIMBP	ECX	Call and return value	
Message	EDX	Return value	
Data	ESI	Return value	
ExID	EDI	Call value	

The function return value (listed below) is set in the AX register.

Success
InvalidExID
NAFnotavailable
NAFBusy
InvalidPCIMPB
InvalidMessageBuffer
PCIMBPTooSmall
MessageBufferTooSmall
MessageTooLarge
DataBufferTooSmall

7.7.1.4.7 PciSetSignal

This function provides a direct information mechanism to be used by the NAF in case of incoming events.

7.7.1.4.7.1 Signal mechanism

7.7.1.4.7.1.1 16 bits PUF

The two mutually exclusive mechanisms described in subclause 7.2.1.4.7 shall be provided by the NAF.

7.7.1.4.7.1.2 32 bits PUF

The two mutually exclusive mechanisms described in subclause 7.2.1.4.7 shall be provided by the 32 bits PCI DLL under Windows 95. The 32 bits PCI DLLs is in charge of the asynchronous signalling when the PCI_CTL_ GETMESSAGE is completed.

To activate signal mechanism, the PUF calls the PciSetSignal function exported by the 32 bits PCI DLL. When the PUF activates this mechanism, the 32 bits PCI DLL shall use the PciGetMessage with the lpOverlapped parameter set in the DeviceIoControl call. Message and data buffers shall be provided by the DLL. The hEvent member of the OVERLAPPED structure specifies the handle of an event that the system sets to the signaled state when the VxD has completed the operation. If DeviceIoControl completes the operation before returning, it returns TRUE otherwise it returns FALSE and sets the extended error information to ERROR_IO_PENDING. When a new message is available, the PciGetMessage is completed and the DLL can access its buffers. Then the 32 bits PCI DLL shall notify the PUF of the new incoming message with the signal mechanism desired.

The PUF calls the PciSetSignal function exported by the 32 bits PCI DLL.

7.7.1.4.7.1.3 VxD PUF

A VxD PUF shall only activate the procedure callback mechanism. The NAF VxD calls the callback function as a near 32 bits function.

The PUF calls the MessageOperations service with VxDCall macro and the following registers:

Name	Parameter	Value	Comment
PUF Version	AH		See table 8 in subclause 5.3.1.4
Function number	AL	0x06	
Signal	ECX	Call value	Always NULL
Signal Procedure	EDX	Call value	
ExID	EDI	Call value	

The function return value (listed below) is set in the AX register

Success
InvalidExID
NAFnotavailable
NAFBusy
InvalidSignalNumber

7.7.1.4.7.2 De activation mechanism

To de-activate any signal mechanism, the PciSetSignal function signal and SignalProcedure parameters shall be NULL. Once de-activated, the previous mechanism shall no longer be used by the NAF to call the PUF.

7.7.1.5 Availability of NAF's PCI_HANDLE

To be accessible via the PciGetHandles function call, a NAF shall issue a declaration action. The inverse action - extraction from the list of available NAFs - is also described. These actions are operating system specific.

7.7.1.5.1 Declaration action

First, the NAF should get the list of available PCI_HANDLES to check if not already declared. If not yet declared, the NAF:

- includes its own PCI_HANDLES into the list of the [Drivers] section of PCI.INI file;
- includes its own PCI_HANDLES into the list of the [Drivers32] section of PCI.INI file;
- includes its unique VxD identifier in the registry.

7.7.1.5.2 Extraction action

First, the NAF may get the list of available PCI_HANDLES to check if already declared. If so, the NAF:

- removes its own PCI_HANDLES from the [Drivers] list in « PCI.INI »;
- removes its own PCI_HANDLES from the [Drivers32] list in « PCI.INI »;
- removes its VxD identifier from the registry.

7.7.2 Windows 95 for Profile B

7.7.2.1 Windows 95 (application level)

Under the operating system Windows 95, three types of user-mode applications can access Profile B:

- DOS based applications;

- Windows 3.x based applications (16-bit);

- Windows 95 based applications (16 bit/32-bit);

Every type of application is able to use Profile B.

Interface Design:

a) **DOS-based applications** continue to use the software interrupt mechanism of Profile B as described in subclause **7.1: MS-DOS**. Additionally a FAR CALL (after pushing flags) to the address of the Profile B shall be supported by an implementation.

b) **Windows based applications** (16-bit) use the DLL mechanism of Profile B as described in subclause **7.2: Windows (application level)** without modifications. The **CAPI20.DLL** provided under Windows 95 has an identical interface to applications.

c) **Windows based applications** (32-bit) can use the DLL mechanism as described in subclause **7.6.2.1: Windows NT (application level)** without modifications. The **CAPI2032.DLL** provided under Windows 95 has the identical interface to applications as in Windows NT.

NOTE: Under Windows 95 the provided DLLs have the same naming convention as in Windows 3.x resp. Windows NT. That implies that a Profile B application written for Windows 3.x also runs under Windows 95.

7.7.2.2 Windows 95 (ODL)

The Profile B for Windows 95 shall be implemented as a Virtual Device Driver (**VxD)**. The interface to such a kernel-mode driver is defined by exported **Virtual Device Services** for other virtual devices and a **Virtual Device API** for protected mode applications (16-bit or 32-bit) accessing the features of the virtual device (i.e. **CAPI20.DLL** / **CAPI2032.DLL**). Both interfaces exchange information by register values. For that, the exchange mechanism in subclause **7.1.2: MS-DOS** is taken and adopted to the 32-bit environment where necessary. The CAPI VxD shall also hook the software interrupt F1 to offer Profile B to DOS based applications.

User-mode applications shall not use the DDL interface directly. Instead they shall use the defined access methods (i.e. software interrupt or DLL mechanism) to access Profile B.

Architectural Overview:

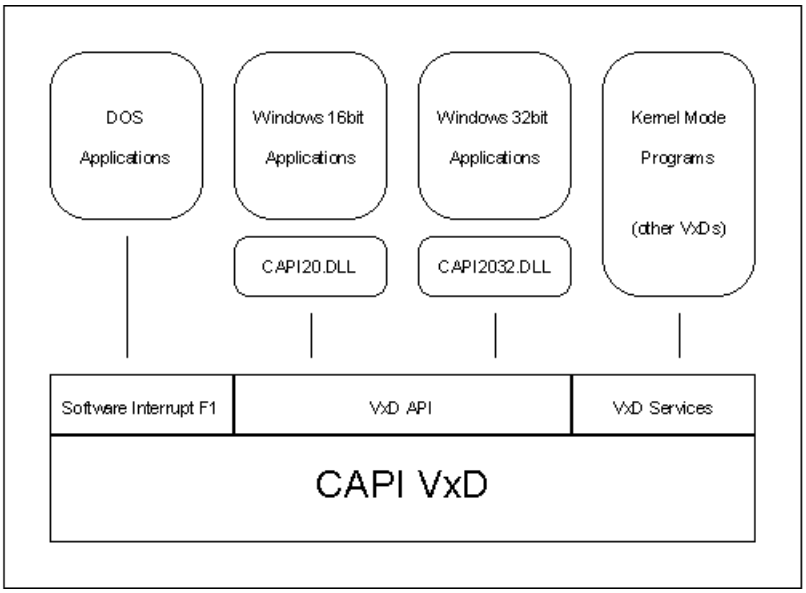


Figure 39: Architectural overview

Virtual Device Services can be used by other virtual devices by including an appropriate include file which contains the service table declaration. A virtual device calls CAPI VxD services using the **VxDcall** macro. To verify the CAPI-VxD services the calling virtual device shall attempt to call the **Get_Version** service of CAPI-VxD. If the CAPI-VxD has not been loaded, the VMM sets the carry flag and returns zero in register **AX**. The virtual device providing Profile B exports *one* Profile B specific service: an access to the message exchange functions described in this subclause. Information is exchanged directly by registers.

The **Virtual Device API** is used by **CAPI20.DLL** and **CAPI2032.DLL**. These DLLs shall retrieve an entrypoint address for this **Virtual Device API** procedure for their virtual machine. The CAPI VxD can obtain the register information of the calling application via the **Client_Reg_Struc** structure.

The CAPI-VxD offers synchrony services. If any Profile B service is entered while an asynchrony interrupt is processed, the return value **0x1107** (internal busy condition) is returned in register **AX**.

Any VxD has a unique device ID. The CAPI-VxD has the device ID **0x3215**.

Service table declaration from CAPI-VxD:

```
VCAPID_DEVICE_ID EQU       3215h
Begin_Service_Table      VCAPID
          VCAPID_Service       VCAPID_Get_Version, LOCAL
          VCAPID_Service       VCAPID_MessageOperations, LOCAL
End_Service_Table        VCAPID
```

In this subclause the term *pointer* refers to the case of the 16 Bit Virtual Device API to a 16:16 (segmented) pointer to a memory location and in case of the 32 Bit Virtual Device API to a 0:32 near flat pointer to a memory location.

7.7.2.2.1 Message operations

7.7.2.2.1.1 CAPI_REGISTER

Description
This is the function the application uses to report its presence to Profile B. By passing the four parameters *ECX*, *EDX*, *ESI* and *EDI* the application describes its needs.

For a "normal" application the size of the message buffer should be calculated using the following formula:

ECX = 1 024 + (1 024 * EDX)

In the *EDX* register the application indicates the maximum number of logical connections opened simultaneously. An attempt to open more logical connections than stipulated here can be acknowledged with an error message from Profile B.

In the *ESI* register the application sets the maximum number of received B3 data blocks that can be reported to the application simultaneously. The number of simultaneously available B3 data blocks has a decisive effect on the throughput of B3 data in the system and should be between 2 and 7. There shall be room for at least two B3 data blocks.

In the *EDI* register the application sets the maximum size of the application data to be transmitted and received, that is the maximum *data length* parameter in messages **DATA_B3_REQ** and **DATA_B3_IND**. The default value for the protocol ISO 7776 [4] (X.75) is 128 octets. Profile B shall support at least up to 2 048 octets, if an application sets register *EDI* with corresponding values.

The application number is supplied in the *AX* register. In the event of an error, the *AX* register is returned with the value 0. The cause of the error is held in the *BX* register in this case.

CAPI_REGISTER	**0x01**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function code 0x01
ECX	Size of message buffer
EDX	Maximum number of level 3 connections
ESI	Number of B3 data blocks available simultaneously
EDI	Maximum size of a B3 data block

Return Value

Return	Value	Comment
AX	<> 0	Application number (ApplID)
	0x0000	Registration error, cause of error in BX register
BX		if AX == 0, coded as described in parameter Info class 0x10xx

NOTE: If the application intends to open a maximum of one layer 3 connection simultaneously and the standard protocols are used, the following register assignment is recommended:
ECX = 2 048, EDX = 1, ESI = 7, EDI = 128
The resulting memory requirement is 2 944 bytes.

7.7.2.2.1.2 CAPI_RELEASE

Description
The application uses this function to log off from Profile B. The memory area indicated in the **CAPI_REGISTER** is released. The application is identified by the application number in the *EDX* register. Any errors that occur are returned in register *AX*.

CAPI_RELEASE	**0x02**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x02
EDX	Application number

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Registration error, coded as described in parameter Info class 0x11xx

7.7.2.2.1.3 CAPI_PUT_MESSAGE

Description
With this function the application transfers a message to Profile B. A pointer is transferred to the message in the *EBX* register. The application is identified via application number in the *EDX* register. Any errors that occur are returned in register *AX*.

CAPI_PUT_MESSAGE	**0x03**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x03
EBX	pointer to the message
EDX	Application number

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
NOTE: After **CAPI_PUT_MESSAGE** the application can use the memory area of the message again. The message shall not be modified by Profile B.		

7.7.2.2.1.4 CAPI_GET_MESSAGE

Description
With this function, the application retrieves a message from Profile B. The application can only retrieve those messages intended for the stipulated application number. A pointer is set to the message in the *EBX* register. If there is no message for the application, the function returns immediately. Register *AX* contains the corresponding error value. The application is identified via the application number in the *EDX* register. Any errors that occur are returned in register *AX*.

CAPI_GET_MESSAGE	**0x04**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x04
EDX	Application number

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
EBX		Pointer to message, if available

NOTE: The message may be invalidated the next time **CAPI_GET_MESSAGE** is called.

7.7.2.2.2 Other functions

7.7.2.2.2.1 CAPI_SET_SIGNAL

Description
The application can use this function to activate usage of the synchron (non-interrupt) call-back function. A pointer to an interrupt call-back function is specified in the *EBX* register. The signalling function can be deactivated by a **CAPI_SET_SIGNAL** with register assignment *EBX* = 0. The application is identified via the application number in the EDX register. Any errors that occurred are returned in the *AX* register.

CAPI_SET_SIGNAL	**0x05**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0x05
EDX	Application number
EDI	Parameter passed to call-back function
EBX	Pointer to call-back function

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

NOTE: The call-back function is always called in a synchron environment, i.e. outside any hardware interrupt condition. It is called as a NEAR function in 32-bit environment, so it shall return by a RET. In case of a usage via the Virtual Device API (i.e. not from another Virtual Device Driver), the context of the calling VM is available.
The call-back function is called by Profile B**,** after
any message is queued in application's message queue
a notified busy condition is cleared
a notified queue full condition is cleared
Interrupts are enabled. The call-back function shall be terminated via RET. All registers shall be preserved.
Profile B shall not call this function recursively. If necessary, the call-back function shall be called again after returning to Profile B.
The call-back function is allowed to use Profile B operations **CAPI_PUT_MESSAGE**, **CAPI_GET_MESSAGE**, and **CAPI_SET_SIGNAL**.
In case of local confirmations (e.g. LISTEN_CONF) the call-back function may be activated before the operation CAPI_PUT_MESSAGE returns to the application.
Parameter EDX and EDI shall be passed to the call-back function with the same values of the corresponding parameters to **CAPI_SET_SIGNAL**.

7.7.2.2.2.2 CAPI_GET_MANUFACTURER

Description
With this function the application determines the manufacturer identification of Profile B**.** In register *EBX* a pointer is transferred to a data area of 64 bytes. The manufacturer identification, coded as a zero terminated ASCII string, is present in this data area after the function has been executed.

CAPI_GET_MANUFACTURER	**0xF0**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF0
EBX	Pointer to buffer
ECX	Number of Controller, if 0 determines the manufacturer identification of the software components

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
EBX		Buffer contains manufacturer identification with ASCII coding. The end of the identification is indicated with a 0 byte.

7.7.2.2.2.3 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

CAPI_GET_VERSION	0xF1

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF1
ECX	Number of Controller, if 0 determines the version of the software components

Return Value

Return	Comment
AH	Profile B major version: 2
AL	Profile B minor version: 0
DH	Manufacturer specific major number
DL	Manufacturer specific minor number

7.7.2.2.2.4 CAPI_GET_SERIAL_NUMBER

Description
With this function the application determines the (optional) serial number of Profile B**.** In register *EBX* a pointer to a data area of 8 bytes is transferred. The serial number, coded as a zero terminated ASCII string, is present in this data area in the form of a seven-digit number after the function has been executed. If no serial number is supplied, the serial number is an empty string.

CAPI_GET_SERIAL_NUMBER	0xF2

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xF2
EBX	Pointer to buffer
ECX	Number of Controller, if 0 determines the serial number of the software components

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx
EBX		The (optional) serial number is read in plain text in the form of a 7-digit number. If no serial number is to be used, a 0 byte shall be written at the first position in the buffer. The end of the serial number is indicated with a 0 byte.

7.7.2.2.2.5 CAPI_GET_PROFILE

Description
The application uses this function to get the capabilities from Profile B. Register *EBX* contain a pointer to a data area of 64 bytes. In this buffer Profile B copies information about implemented features, number of controllers and supported protocols. Register *ECX* contains the controller number (bit 0..6) for which this information is requested.

CAPI_GET_PROFILE	**0xF3**

Parameter	Comment
AH	Version number 20 (0x14)
AL	Functional Code 0xF3
ECX	controller number (if 0, only number of controllers is returned)
EBX	pointer to buffer

Return Value

Return	Value	Comment
AX	0x0000	No error
	<> 0	Coded as described in parameter info class 0x11xx

Retrieved structure format:

Type	Description
2 octets	number of installed controllers, least significant octet first
2 octets	number of supported B-channels, least significant octet first
4 octets	Global Options (bit field): 0: internal controller supported 1: external equipment supported 2: Handset supported (external equipment shall be set also) 3: DTMF supported 4..31: reserved
4 octets	B1 protocols support (bit field): 0: 64 kBit/s with HDLC framing, always set. 1: 64 kBit/s bit transparent operation with byte framing from the network 2: V.110 [17] asynchronous operation with start/stop byte framing 3: V.110 [17] synchronous operation with HDLC framing 4: T.30 [14] modem for facsimile group 3 5: 64 kBit/s inverted with HDLC framing. 6: 56 kBit/s bit transparent operation with byte framing from the network 7..31: reserved
4 octets	B2 protocol support (bit field): 0: ISO 7776 [4] (X.75 SLP), always set 1: Transparent 2: SDLC [12] 3: LAPD according Q.921 [13] for D-channel X.25 4: T.30 [14] for facsimile group 3 5: Point to Point Protocol (PPP [10] [11]) 6: Transparent (ignoring framing errors of B1 protocol) 7..31: reserved
4 octets	B3 protocol support (bit field): 0: Transparent, always set 1: T.90NL with compatibility to T.70NL according to T.90 Appendix II [16]. 2: ISO 8208 [3] (X.25 DTE-DTE) 3: X.25 DCE 4: T.30 [14] for facsimile group 3 5..31: reserved
24 octets	reserved for Profile B usage
20 octets	manufacturer specific information
NOTE: This function can be extended, so an application has to ignore unknown bits. Profile B shall set every reserved field to 0.	

7.7.2.2.2.6 CAPI_MANUFACTURER

Description
This function is manufacturer specific.

CAPI_MANUFACTURER	0xFF

Parameter	Comment
AH	Version number 20 (0x14)
AL	Function Code 0xFF
Manufacturer specific	

Return Value

Return	Comment
Manufacturer specific	

7.7.2.3 Windows 95 (DeviceIoControl)

Profile B can also be accessed by using DeviceIoControl operations. The definition of this interface is as close as possible to the definition of the Windows NT DeviceIoControl interface. Since not all Windows NT device operations are available under Windows 95 this interface cannot be defined as being compatible to the Windows NT definition.

The following DEVICE_CONTROL codes are defined for the Profile B functions:

```
/*
*       the common device type code for CAPI20 conforming drivers
*/
#define FILE_DEVICE_CAPI20 0x8001

/*
*       DEVICE_CONTROL codes
*/
#define CAPI_CTL_BASE                   0x800
#define CAPI_CTL_REGISTER               (CAPI_CTL_BASE+0x0001)
#define CAPI_CTL_RELEASE                (CAPI_CTL_BASE+0x0002)
#define CAPI_CTL_PUT_MESSAGE            (CAPI_CTL_BASE+0x0003)
#define CAPI_CTL_GET_MESSAGE            (CAPI_CTL_BASE+0x0004)
#define CAPI_CTL_GET_MANUFACTURER       (CAPI_CTL_BASE+0x0005)
#define CAPI_CTL_GET_VERSION            (CAPI_CTL_BASE+0x0006)
#define CAPI_CTL_GET_SERIAL             (CAPI_CTL_BASE+0x0007)
#define CAPI_CTL_GET_PROFILE            (CAPI_CTL_BASE+0x0008)
#define CAPI_CTL_WAIT_MESSAGE           (CAPI_CTL_BASE+0x0009)
#define CAPI_CTL_MANUFACTURER           (CAPI_CTL_BASE+0x00ff)


/*
*       The wrapped control codes as required by the system. Note: while use of these macros is not
        required,
*       no other control parameters are allowed for the DeviceIoControl control codes.
*/
#define CAPI_CTL_CODE(function,method) \
            CTL_CODE(FILE_DEVICE_CAPI20,function,method,FILE_ANY_ACCESS)

#define IOCTL_CAPI_REGISTER \
        CAPI_CTL_CODE(CAPI_CTL_REGISTER, METHOD_BUFFERED)

#define IOCTL_CAPI_RELEASE \
        CAPI_CTL_CODE(CAPI_CTL_RELEASE, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_MANUFACTURER\
        CAPI_CTL_CODE(CAPI_CTL_GET_MANUFACTURER, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_VERSION \
        CAPI_CTL_CODE(CAPI_CTL_GET_VERSION, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_SERIAL \
        CAPI_CTL_CODE(CAPI_CTL_GET_SERIAL, METHOD_BUFFERED)

#define IOCTL_CAPI_GET_PROFILE \
        CAPI_CTL_CODE(CAPI_CTL_GET_PROFILE, METHOD_BUFFERED)

#define IOCTL_CAPI_MANUFACTURER \
        CAPI_CTL_CODE(CAPI_CTL_MANUFACTURER, METHOD_BUFFERED)
```

```
#define IOCTL_CAPI_PUT_MESSAGE \
        CAPI_CTL_CODE(CAPI_CTL_PUT_MESSAGE, METHOD_IN_DIRECT)

#define IOCTL_CAPI_GET_MESSAGE \
        CAPI_CTL_CODE(CAPI_CTL_GET_MESSAGE, METHOD_OUT_DIRECT)

#define IOCTL_CAPI_WAIT_MESSAGE \
        CAPI_CTL_CODE(CAPI_CTL_WAIT_MESSAGE, METHOD_BUFFERED)
```

CAPI20 specific return values are mapped to Win32 error codes according to the following table. The error code is returned by GetLastError() after failure of DeviceIoControl().

Info	Win32 Error Code
0x1001	ERROR_TOO_MANY_SESSIONS
0x1002	ERROR_INVALID_PARAMETER
0x1003	
0x1004	ERROR_INSUFFICIENT_BUFFER
0x1005	ERROR_NOT_SUPPORTED
0x1006	
0x1007	ERROR_NETWORK_BUSY
0x1008	ERROR_NOT_ENOUGH_MEMORY
0x1009	
0x100a	ERROR_SERVER_DISABLED
0x100b	ERROR_SERVER_NOT_DISABLED
0x1101	ERROR_INVALID_HANDLE
0x1102	ERROR_INVALID_FUNCTION
0x1103	ERROR_TOO_MANY_CMDS
0x1104	ERROR_IO_PENDING
0x1105	ERROR_IO_DEVICE
0x1106	STATUS_INVALID_PARAMETER
0x1107	ERROR_BUSY
0x1108	ERROR_NOT_ENOUGH_MEMORY
0x1109	
0x110a	ERROR_SERVER_DISABLED
0x110b	ERROR_SERVER_NOT_DISABLED

In Windows 95 all communications between a CAPI20 device and an application is related to a file handle. For that reason a file handle is used to identify a link between a CAPI20 device and the application instead of the application id. Any application ids within Profile B messages are therefore ignored.

In the following the interface between the application and the Profile B device driver is described with Win32 functions.

7.7.2.3.1 Message operations

7.7.2.3.1.1 CAPI_REGISTER

Description
This is the operation the application uses to report its presence to Profile B. By passing the four parameters MessageBufferSize, maxLogicalConnection, maxBDataBlocks and maxBDataLen the application describes its needs.

For a 'normal' application the size of the message buffer should be calculated using following formula:

MessageBufferSize = 1 024 + (1 024 * maxLogicalConnection)

CAPI_REGISTER	CAPI_CTL_REGISTER

Implementation
For the CAPI_REGISTER function the application shall get a handle to the Profile B device using the Win32 CreateFile() function and then send a CAPI_CTL_REGISTER to the Profile B device. With CAPI_REGISTER the following data structure is passed on to the driver:

```
struct capi_register_params {
        WORD MessageBufferSize,
        WORD maxLogicalConnection,
        WORD maxBDataBlocks,
        WORD maxBDataLen
};
```

Only one CAPI_CTL_REGISTER may be sent with one handle if no CAPI_CTL_RELEASE is sent in between. If an application program wants to register as more than one Profile B application, it shall obtain several handles with CreateFile() and send one CAPI_CTL_REGISTER with each handle. The FILE_FLAG_OVERLAPPED option for fdwAttrsAndFlags shall be set for proper operation.

EXAMPLE:

```
capi_handle = CreateFile("\\\\.\\CAPI20",
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
                NULL);

r.MessageBufferSize = MessageBufferSize;
r.maxLogicalConnections = maxLogicalConnections;
r.maxBDataBlock = maxBDataBlocks;
r.maxBDataLen = maxBDataLen;

ret = DeviceIoControl(capi_handle,
            CAPI_CTL_REGISTER,
            (PVOID) &r,
            sizeof(r),
            NULL,
            0,
            &ret_bytes,
            NULL);
```

7.7.2.3.1.2 CAPI_RELEASE

Description
The application uses this operation to log off from Profile B. This way the application signals Profile B that all resources that have been allocated by Profile B for the application can be released again.

CAPI_RELEASE	CAPI_CTL_RELEASE

Implementation
A CAPI_RELEASE can be performed in two ways. If the same handle is to be used again, an CAPI_CTL_RELEASE shall be sent. If the handle is not used any more the handle may just be closed using CloseHandle.

EXAMPLE:

```
ret = DeviceIoControl(capi_handle,
                CAPI_CTL_RELEASE,
                NULL,
                0,
                NULL,
                0,
                &ret_bytes,
                NULL);
CloseHandle(capi_handle);
```

7.7.2.3.1.3 CAPI_PUT_MESSAGE

Description
With this operation the application transfers a message to Profile B. The application identifies itself with an file handle.

CAPI_PUT_MESSAGE	**CAPI_CTL_PUT_MESSAGE**

Implementation
The CAPI_PUT_MESSAGE function is performed using a CAPI_CTL_PUT_MESSAGE DeviceIoControl.

With this DeviceIoControl operation one data buffer is sent to the CAPI20 device driver. This buffer shall contain the message AND data associated with a DATA_B3_REQ message. The data (if available) shall be placed into the buffer immediately following the message.

```
ret = DeviceIoControl(capi_handle,
                CAPI_CTL_PUT_MESSAGE
                (PVOID)msg,             /* buffer for message + data */
                msg_length,                     /* length of message + data */
                NULL,
                0,
                &ret_bytes,
                NULL);
```

The operation completes immediately, it does not wait for any network event (normal CAPI_PUT_MESSAGE operation).

The buffer can be re-used by the application, as soon as the operation completes.

7.7.2.3.1.4 CAPI_GET_MESSAGE

Description
With this operations the application retrieves a message from Profile B. The application retrieves all messages associated with the corresponding file handle from operation **CAPI_REGISTER**.

CAPI_GET_MESSAGE	**CAPI_CTL_GET_MESSAGE**

Implementation
The CAPI_GET_MESSAGE function is performed using the CAPI_CTL_GET_MESSAGE DeviceIoControl operation.

With the CAPI_CTL_GET_MESSAGE DeviceIoControl operation one data buffer is received from the CAPI20 device driver. This buffer contains the message AND data associated with a DATA_B3_IND message. The data (if available) is placed into the buffer immediately following the message.

The CAPI_CTL_GET_MESSAGE supports overlapped operation. If it returns TRUE, the number of bytes of the message retrieved is available.

If the buffer provided by the application is to small to hold the message and the data, an error ERROR_UNSUFFICIENT_BUFFER shall be returned and no message is retrieved.

EXAMPLE:

```
ret = DeviceIoControl(capi_handle,
                CAPI_CTL_GET_MESSAGE,
                NULL,
                0,
                (PVOID)buffer,    /* buffer for message + data */
                buffer_size,              /* length of message + data */
                &ret_bytes,
                &0_read);

if (ret == TRUE) {
        /* operation immediately succeeded */
        /* ret_bytes contains the number of bytes accepted */
            ;
} else if (GetLastError() == ERROR_IO_PENDING) {
        /* operation pending, must wait for completion */
        WaitForSingleObject(result.hEvent, INFINITE);
        ret = GetOverlappedResult(capi_handle, &result, &ret_bytes, TRUE);
        if (ret == TRUE) {
            /* operation successful completed now */
            /* ret_bytes contains the number of bytes accepted */
                ;
        } else {
            /* sorry, failure */
            ...
        }
} else {
        /* operation immediately failed */
        ...
}
```

7.7.2.3.1.5 CAPI_SET_SIGNAL

Description
There is no CAPI_SET_SIGNAL function. The asynchronous signalling of a received message is done implicitly by completing the CAPI_CTL_GET_MESSAGE operation.

7.7.2.3.2 Other functions

7.7.2.3.2.1 CAPI_GET_MANUFACTURER

Description
With this operation the application determines the manufacturer identification of Profile B. The offered buffer shall have a size of at least 64 bytes. Profile B copies the identification string, coded as a zero terminated ASCII string, to this buffer.

CAPI_GET_MANUFACTURER	CAPI_CTL_GET_MANUFACTURER

Implementation
With this CAPI_CTL_GET_MANUFACTURER the manufacturer identification is read from the Profile B driver. A buffer of 64 bytes shall be provided by the application. The manufacturer identification is returned as zero terminated ASCII string. Controller number 0 retrieves the manufacturer name of the CAPI20 device driver, other controller numbers retrieve the manufacturer of the corresponding controller.

```
DWORD controller;               /* 32 bit */
char    manufacturer[64];

controller = 0;                 /* to retrieve the manufacturer of the device driver */
ret = DeviceIoControl(capi_handle,
            CAPI_CTL_GET_MANUFACTURER,
            (PVOID) &controller,
            sizeof (controller),
            (PVOID) manufacturer,
            sizeof (manufacturer),
            &ret_bytes,
            (POVERLAPPED) NULL);
```

7.7.2.3.2.2 CAPI_GET_VERSION

Description
With this function the application determines the version of Profile B as well as an internal revision number.

CAPI_GET_VERSION	CAPI_CTL_GET_VERSION

Implementation
With this CAPI_CTL_GET_VERSION the version of the Profile B can be read. A buffer with the following structure shall be provided by the application: Controller number 0 retrieves the version info of the CAPI20 device driver, other controller numbers retrieve the version of the corresponding controller.

```
struct capi_version_params {
            WORD CAPIMajor;         /* 16 bit */
            WORD CAPIMinor;
            WORD ManufacturerMajor;
            WORD ManufacturerMinor;
} buf;
DWORD controller;               /* 32 bit */

controller = 0;                 /* to retrieve the version of the device driver */
ret = DeviceIoControl(capi_handle,
            CAPI_CTL_GET_VERSION,
            (PVOID) &controller,
            sizeof (controller),
            (PVOID) &buf,
            sizeof (buf),
            &ret_bytes,
            (POVERLAPPED) NULL);
```

7.7.2.3.2.3 CAPI_GET_SERIAL_NUMBER

Description
With this operation the application determines the (optional) serial number of Profile B. The offered buffer shall have a size of 8 bytes. Profile B copies the serial number string to this buffer. The serial number, coded as a zero terminated ASCII string, represents seven digit number after the function has returned.

CAPI_GET_SERIAL_NUMBER	CAPI_CTL_GET_SERIAL_NUMBER

Implementation
With this CAPI_CTL_GET_SERIAL_NUMBER the Profile B serial number can be read from the driver. A buffer of 8 bytes shall be provided by the application. The serial number is returned as zero terminated ASCII string. Controller number 0 retrieves the serial number of the CAPI20 device driver, other controller numbers retrieve the serial number of the corresponding controller.

```
char   serial[8];
DWORD     controller;          /* 32 bit */

controller = 0;                        /* to retrieve the serial number of the device driver */
ret = DeviceIoControl(capi_handle,
          CAPI_CTL_GET_SERIAL_NUMBER,
          (PVOID) &controller,
          sizeof (controller),
          (PVOID) serial,
          sizeof (serial),
          &ret_bytes,
          (POVERLAPPED) NULL);
```

7.7.2.3.2.4 CAPI_GET_PROFILE

Description
With this DeviceIoControl the Profile B capabilities can be read from the driver. A buffer of 64 bytes shall be provided by the application. The same profile structure as for the Windows 95 application level 32 bit DLL interface is returned by the driver.

CAPI_GET_PROFILE	CAPI_CTL_GET_PROFILE

Implementation
With this DeviceIoControl the Profile B capabilities can be read from the driver. A buffer formatted according to the CAPI profile structure shall be provided by the application in the buffer parameter. It shall be filled with the appropriate values by the DeviceIoControl call.

```
char       profile[64];
DWORD controller;          /* 32 bit */

controller = 1;                /* to retrieve the profile of controller number one*/
ret = DeviceIoControl(capi_handle,
          CAPI_CTL_GET_PROFILE,
          (PVOID) &controller,
          sizeof (controller),
          (PVOID)profile,
          sizeof (profile),
          &ret_bytes,
          (POVERLAPPED) NULL);
```

Annex A (informative): Bibliography

This bibliography contains references to documents which are of importance to the PUF and NAF developers. The documents can be useful when reading or implementing this ETS.

Directive 86/659/EEC: "Council recommendation of 22 December 1986 on the coordinated introduction of the integrated services digital network (ISDN) in the European Community".

ETR 018 (1991): "Integrated Services Digital Network (ISDN); Application of the BC-,HLC-, LLC-information elements by terminals supporting ISDN services".

ETS 300 050 (1991): "Integrated Services Digital Network (ISDN); Multiple Subscriber Number (MSN) supplementary service Service description".

ETS 300 051 (1991): "Integrated Services Digital Network (ISDN); Multiple Subscriber Number (MSN) supplementary service Functional capabilities and information flows".

ETS 300 052 (1991): "Integrated Services Digital Network (ISDN); Multiple Subscriber Number (MSN) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 053 (1991): "Integrated Services Digital Network (ISDN); Terminal Portability (TP) supplementary service Service description".

ETS 300 054 (1991): "Integrated Services Digital Network (ISDN); Terminal Portability (TP) supplementary service Functional capabilities and information flows".

ETS 300 055 (1991): "Integrated Services Digital Network (ISDN); Terminal Portability (TP) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 056 (1991): "Integrated Services Digital Network (ISDN); Call Waiting (CW) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 057 (1992): "Integrated Services Digital Network (ISDN); Call Waiting (CW) supplementary service description".

ETS 300 058 (1991): "Integrated Services Digital Network (ISDN); Call Waiting (CW) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 059 (1991): "Integrated Services Digital Network (ISDN); Subaddressing (SUB) supplementary service Service description".

ETS 300 060 (1991): "Integrated Services Digital Network (ISDN); Subaddressing (SUB) supplementary service Functional capabilities and information flows".

ETS 300 061 (1991): "Integrated Services Digital Network (ISDN); Subaddressing (SUB) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol"..

ETS 300 062 (1991): "Integrated Services Digital Network (ISDN); Direct Dial In (DDI) supplementary service Service description".

ETS 300 063 (1991):"Integrated Services Digital Network (ISDN); Direct Dial In (DDI) supplementary service Functional capabilities and information flows".

ETS 300 064 (1991): "Integrated Services Digital Network (ISDN); Direct Dial In (DDI) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 089 (1992): "Integrated Services Digital Network (ISDN); Calling Line Identification Presentation (CLIP) supplementary service Service description".

ETS 300 090 (1992): "Integrated Services Digital Network (ISDN); Calling Line Identification Restriction (CLIR) supplementary service Service description".

ETS 300 091 (1992): "Integrated Services Digital Network (ISDN); Calling Line Identification Presentation (CLIP) and Calling Line Identification Restriction (CLIR) supplementary services Functional capabilities and information flows".

ETS 300 092 (1992): "Integrated Services Digital Network (ISDN); Calling Line Identification Presentation (CLIP) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 093 (1992): "Integrated Services Digital Network (ISDN); Calling Line Identification Restriction (CLIR) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 102-2 (1990): "Integrated Services Digital Network (ISDN); User-network interface layer 3 Specification for basic call control Specification Description Language (SDL) diagrams".

ETS 300 179: "Integrated Services Digital Network (ISDN); Advice of Charge: charging information during the call (AOC-D) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 180: "Integrated Services Digital Network (ISDN); Advice of Charge: charging information at the end of the call (AOC-E) supplementary service Digital Subscriber Signalling System No. one (DSS1) protocol".

ETS 300 181: "Integrated Services Digital Network (ISDN); Advice of Charge (AOC) supplementary service description".

ETS 300 182: "Integrated Services Digital Network (ISDN); Advice of Charge (AOC) Digital Subscriber Signalling System No. one (DSS1) protocol".

ISO 8878 (1990): "Information processing systems - Data communications - Use of X.25 to provide the OSI connection-mode network service".

ISO/IEC 9574 (1989): "Information technology - Telecommunications and information exchange between systems - Provision of the OSI connection-mode network service by packet mode terminal equipment connected to an integrated services digital network (ISDN)".ITU-T Recommendation F.721 (08/92): “Videotelephony teleservice for ISDN”.

ITU-T Recommendation H.221 (1993): “Frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices”.

CCITT Recommendation X.21 (09/92): “Interface between data terminal equipment and data circuit-terminating equipment for synchronous operation on public data networks”.

CCITT Recommendation X.211 (1988): "Physical Service Definition for Open Systems Interconnection for CCITT Applications".ITU-T Recommendation X.200 (07/94): "Information technology - Open Systems Interconnection - Basic reference model: The basic model".

ITU-T Recommendation X.400 (1993): "Message handling services: Message handling system and service overview".

Annex B (normative): Mapping between Profile A messages and parameters and the ISDN

This annex provides the mapping between protocols used and the Profile A messages.

B.1 Control Plane messages

Table B.1: Control Plane message to ETS 300 102 [2] mapping

PCI Message	ETS 300 102 [2] Message	Direction	NOTES
CAlertReq	Alerting	user to network	
CAlertInd	Alerting	network to user	
CConnectReq	Setup	user to network	
CConnectInd	Setup	network to user	
CConnectRsp	Connect	user to network	
CConnectCnf	Connect	network to user	
CDisconnectReq	Disconnect, Release, Release Complete	user to network	note 1
CDisconnectInd	Disconnect, Release, Release Complete	network to user	note 1
CDisconnectRsp	Release	user to network	note 1
CDisconnectCnf	Release, Release Complete	network to user	note 1
CProgressInd	Progress	network to user	
CStatusInd	Status	network to user	note 2
CProceedingInd	Call proceeding	network to user	
CSetupAckInd	Setup acknowledge	network to user	
CConnectInfoReq	Information	user to network	
CUserInformationReq	User Information	user to network	
CUserInformationInd	User Information	network to user	
CCongestionControlReq	Congestion Control	user to network	
CCongestionControlInd	Congestion control	network to user	
CSuspendReq	Suspend	user to network	
CSuspendCnf	Suspend acknowledge, Suspend reject	network to user	
CResumeReq	Resume	user to network	
CResumeCnf	Resume acknowledge, Resume reject	network to user	
CNotifyInd	Notify	network to user	
CFacilityReq	Facility	user to network	
CFacilityInd	Facility	network to user	
CAddInfoReq	Information	user to network	
CAddInfoInd	Information	network to user	
CDtmfReq	In band message without relationship with signalling	user to network	
CDtmfCnf	In band message without relationship with signalling	network to user	
CDtmfInd	In band message without relationship with signalling	network to user	

NOTE 1: In the case of the PCI CDisconnect* messages the specific message received or sent to the ISDN depends upon the state of the call when the CDisconnect* message is received from or sent to the PUF. Depending on the ISDN message that caused the CDisconnectInd, CDisconnectRsp may or may not cause a message to be sent to the ISDN. CDisconnectCnf shall not be mapped from a message from the ISDN when CDisconnectReq is used to respond to CConnectInd.

NOTE 2: This PCI message may be generated by a protocol error detected by the NAF or by a protocol error indicated by a status message received from the ISDN.

NOTE 3: External equipment messages are not included in this table.

B.2 Control Plane parameters

The mapping of Control Plane parameters to the ETS 300 102 [2] information elements is defined in table B.2.

Table B.2: Control Plane parameters

Control Plane parameter	Q.931 [2] Information Element
BearerCap	Bearer Capability
CalledNumber	Called party number
CalledSubaddress	Called party subaddress
CallingNumber	Calling party number
CallingSubaddress	Calling party subaddress
CauseToPUF	Cause
CauseToNAF	Cause
ChannelIdentification	Channel Identification
CongestionLevel	Congestion level
ConnectedNumber	Called party number
ConnectedSubaddress	Called party subaddress
ControllerID	No relationship
DateTime	Date/time
Display	Display
DtmfOperation	No relationship
DtmfToneDuration	No relationship
DtmfGapDuration	No relationship
DtmfDigits	No relationship
DtmfResult	No relationship
Facility	Facility
HLC	High layer compatibility
Keypad	Keypad facility
LLC	Low layer compatibility
NotificationIndicator	Notification Indicator
NumberComplete	Sending complete
ProgressIndicator	Progress Indicator
Signal	Signal
UserToUserInfo	User-user

Annex C (normative): Telephony defined in the Profile A

This annex presents different types of external equipment handled in this ETS.

C.1 Type 1 external equipment

This external equipment is the simplest form of telephony equipment. It does not contain hook control or a dialling mechanism. It only contains the transceivers and does not manage the ISDN signalling. It is totally under the control of the NAF. A Control Plane message is defined to indicate to the PUF the availability of the external equipment (external equipment connected or not to the NAF).

It is the responsibility of the NAF to connect a channel to this type of external equipment when the channel becomes active.

If the external equipment is in use, a CConnectReq that attempts to use the external equipment should be rejected with a CDisconnectInd with Cause value 47 (Resource unavailable).

If the external equipment is in use, and an incoming call arrives that attempts to use the external equipment, the NAF should pass a CConnectInd to the relevant PUF. The PUF is then in control to make the external equipment available for use. If it does not, an attempt to connect shall be denied with CDisconnectInd with Cause value 47 (resource unavailable).

C.2 Type 2 external equipment

This external equipment contains hook control but not a dialling mechanism. This external equipment does not manage ISDN signalling. It can provide some information to the PUF about the state of the handset by the means of two Control Plane messages. Therefore, this external equipment can cause state transitions in the PUF for incoming and outgoing calls.

A Control Plane message is defined to indicate to the PUF the availability of the external equipment (external equipment connected or not to the NAF).

It is the responsibility of the NAF to connect a channel to this type of external equipment when the channel becomes active. It is the responsibility of the PUF to ensure that the hook control is in the desired state when the channel becomes active.

If the equipment is in use, and an incoming call arrives that attempts to use the equipment, the NAF should pass a CConnectInd to the relevant PUF. The PUF is then in control to make the external equipment available for use. If it does not, an attempt to connect shall be denied with CDisconnectInd with cause 47 (Resource unavailable).

C.3 Type 3 external equipment

This contains hook control but not a dialling mechanism. This external equipment is connected to the ISDN network; therefore, it is able to manage ISDN signalling when an incoming call arrives in the case where the host is off.

It can provide some information to the PUF about the state of the handset by the means of two Control Plane messages. Therefore, this external equipment can cause state transitions in the PUF for incoming and outgoing calls.

A Control Plane message is defined to indicate to the PUF the availability of the external equipment (external equipment connected or not to the NAF).

If the external equipment is in use, and an incoming call arrives that attempts to use the equipment, the NAF should pass a CConnectInd to the relevant PUF. The PUF then determines whether or not to make the external equipment available for use. If it does not, an attempt to connect shall be denied with CDisconnectInd with cause 47 (Resource unavailable).

C.4 Type 4 external equipment

This external equipment contains a dialling mechanism with or without a hook control. This external equipment does not manage ISDN signalling. This kind of external equipment supports dialling with block sending or overlap sending. In the case of an overlap sending, a Control Plane message containing the code of the key pressed on the keypad, per key pressed, is provided to the PUF. In the case of a block sending, a single Control Plane message containing the complete remote address and/or subaddress is provided to the PUF.

If this external equipment contains a hook control, it can provide some information to the PUF about the state of the handset by the means of two Control Plane messages.

A Control Plane message is defined to indicate to the PUF the availability of the external equipment (external equipment connected or not to the NAF).

All dialling actions and handset actions (if available) realised on this external equipment can cause state transition in the PUF for incoming and outgoing calls.

It is the responsibility of the NAF to connect a channel to this type of external equipment when the channel becomes active.

If the equipment is in use, and an incoming call arrives that attempts to use the equipment, the NAF should pass a CConnectInd to the relevant PUF. The PUF is then determines whether or not to make the external equipment available for use. If it does not, an attempt to connect shall be denied with CDisconnectInd with cause 47 (Resource unavailable).

C.5 Type 5 external equipment

This external equipment contains a dialling mechanism with or without a hook control. This external equipment is connected to the ISDN network; therefore, it is able to manage ISDN signalling in the case where the host (e.g. personal computer) is off. Type 5 external equipment allows placing outgoing calls from its and answering incoming calls.

This kind of external equipment can allow dialling with block sending or overlap sending. In the case of an overlap sending, a Control Plane message containing the code of the key pressed on the keypad, per key pressed, is provided to the PUF. In the case of a block sending, a single Control Plane message containing the complete remote address and/or subaddress is provided to the PUF.

If this external equipment contains hook control, it can provide some information to the PUF about the state of the handset by the means of two Control Plane messages.

A Control Plane message is defined to indicate to the PUF the availability of the external equipment (external equipment connected or not to the NAF).

All Type 5 dialling actions and handset operations (if available) can cause state transitions in the PUF for incoming and outgoing calls.

It is the responsibility of the NAF to connect a channel to this type of external equipment when the channel becomes active.

If the equipment is in use, and an incoming call arrives that attempts to use the equipment, the NAF should pass a CConnectInd to the relevant PUF. The PUF then determines whether or not to make the external equipment available for use. If it does not, an attempt to connect shall be denied with CDisconnectInd with cause 47 (Resource unavailable).

Annex D (normative): X.25 usage in the Profile A

D.1 Parameter Values for ITU-T Recommendation X.25 use

Table D.1 shows the required setting of parameters to achieve different types of ITU-T Recommendation X.31 operation.

Table D.1: Types of ITU-T Recommendation X.31 operation

Type of X.31 Operation	BearerCap	Channel Selection	Channel Number	Called Number
X.31 Case A Switched	64 KHZ	Not Required	Not Required	Required
X.31 Case A Permanent	64 KHZ	B-channel	Required	Not Required
X.31 Case B, B-channel switched	X.25	Not Required	Not Required	Not Required
X.31 Case B, B-channel permanent	X.25	B-channel	Required	Not Required
X.31 Case B, D-channel	X.25	D-channel	Required	Not Required

D.2 Disconnection of ISDN channel with established X.25 Connections

In the co-ordination case, this is covered by ISO 9574 [9].

In the non-co-ordination case, the following should be provided to the PUF:

- CDisconnectInd message with cause for channel disconnection;
- for each established ITU-T Recommendation X.25 Connection:
  - UDisconnectInd message with X213Cause and X213Origin as defined by ISO 9574 [9] and X25Cause.

- for each ITU-T Recommendation X.25 Connection in the process of being established:
  - UDisconnectInd message with X213Cause and X213Origin as defined by ISO 9574 [9] and X25Cause.

Annex E (Informative): Profile A NAF development guidelines

The main body of this ETS which deals with the Profile A contains the description of the Profile A from the PUF point of view. Following this approach, certain points, not directly related to the PUF, which have an impact on the development of the NAF are not described. These points may be of interest for the NAF development and are, therefore, described in this annex. It gives guidelines for the development of the NAF in accordance with the main body of this ETS.

An example of a point which is not covered in the main body is the mapping between the coding for the AOC supplementary service and the special coding used in Profile A.

Consider this annex from the following viewpoints:

- this annex gives additional points. The NAF is implemented using this ETS. It should implement the Profile A in such a way that the functionality described is provided;

- the main body of this ETS should be given priority if there is anything not clear in this annex or the interpretation between the main body of this ETS and this annex is different;

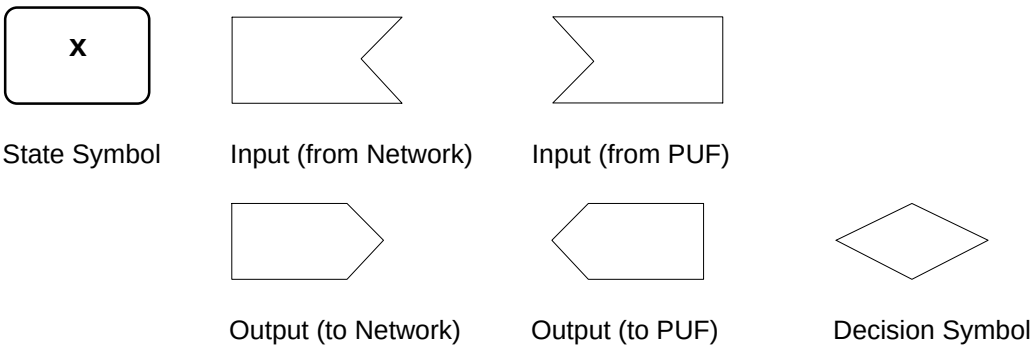
- this annex does not try to impose any constraints on the implementation of the NAF. The objective is to give guidelines as to how the NAF can be developed to be in line with this ETS.

E.1 NAF SDL diagrams

E.1.1 NAF SDL diagrams: conventions

The mapping of Control or User Plane messages to protocol messages is provided in the following tables. Some SDL diagrams, or other kind of schemes, are given to explain more clearly the relation between user messages and network primitives. These diagrams do not cover every case. They only present some of the possible cases.

The following symbols are used within this description. A full description of the symbols and their meaning is given in ITU-T Recommendation Z.100 [5].



E.1.2 NAF SDL diagrams for Control Plane

The following SDL diagrams show, as an example, the internal states of the call control section of the NAF. They are provided for clarification only. Not all the possible cases are shown in these diagrams, for simplification.

The primitives shown in upper case are those defined in ETS 300 102 [2].

The following symbols are used within this description. A full description of the symbols and their meaning is given in ITU-T Recommendation Z.100 [5].

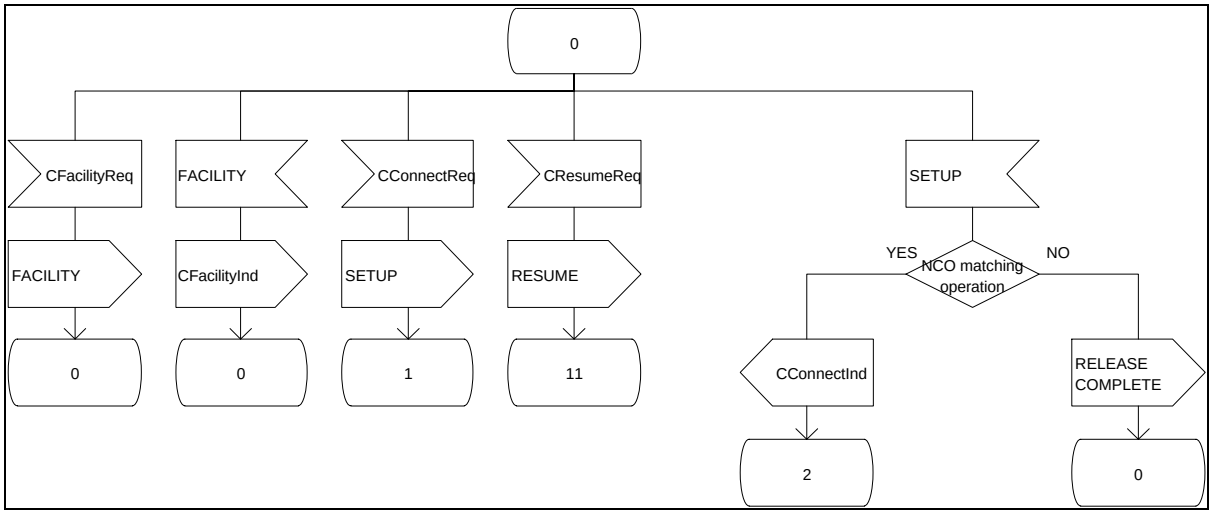


Figure E.1: IDLE

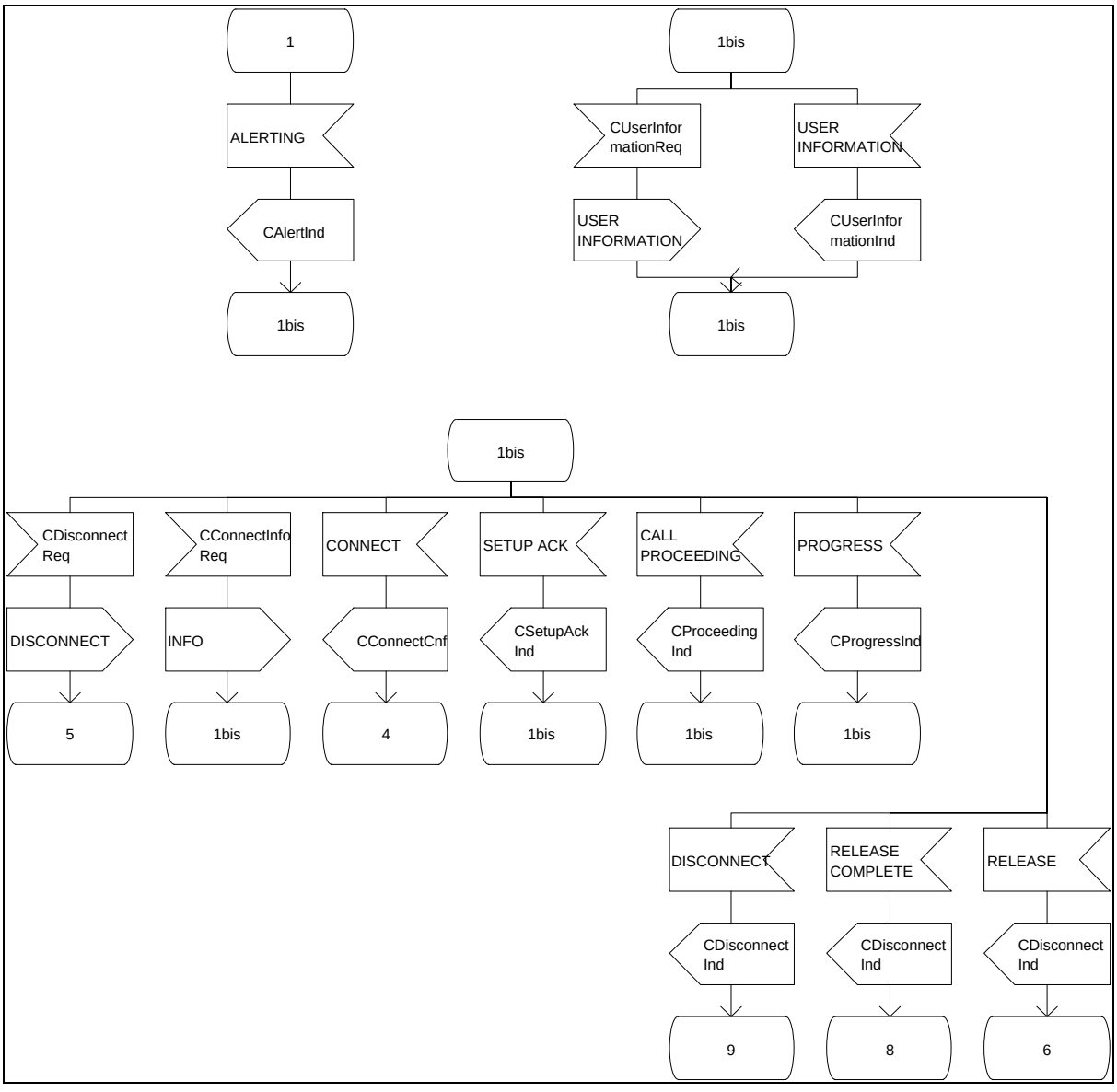


Figure E.2: CALL INITIATED

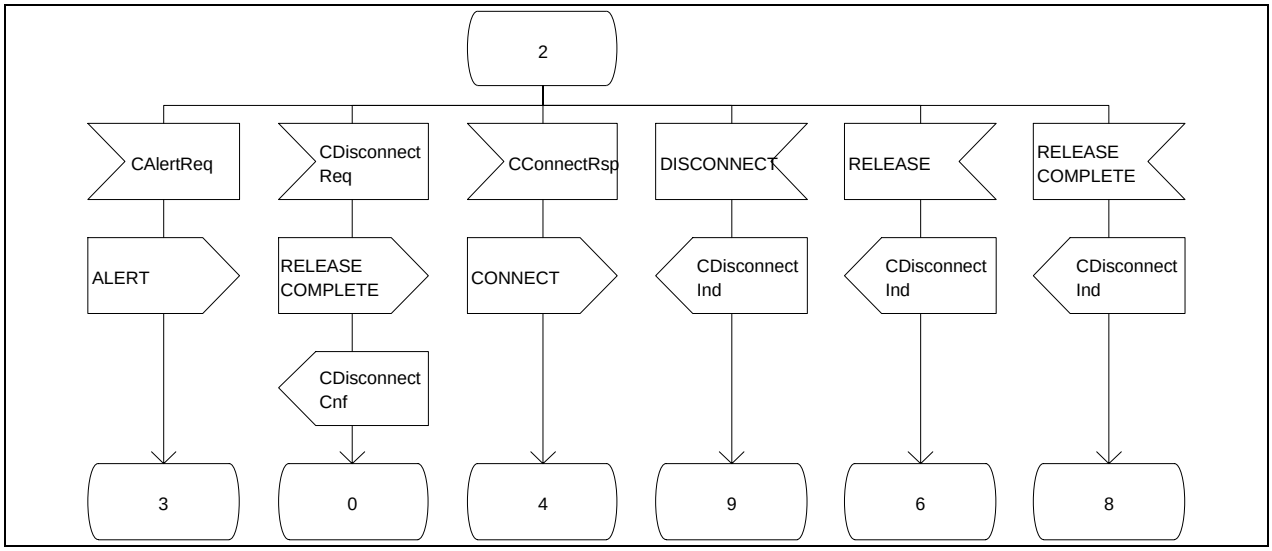


Figure E.3: CALL PRESENT

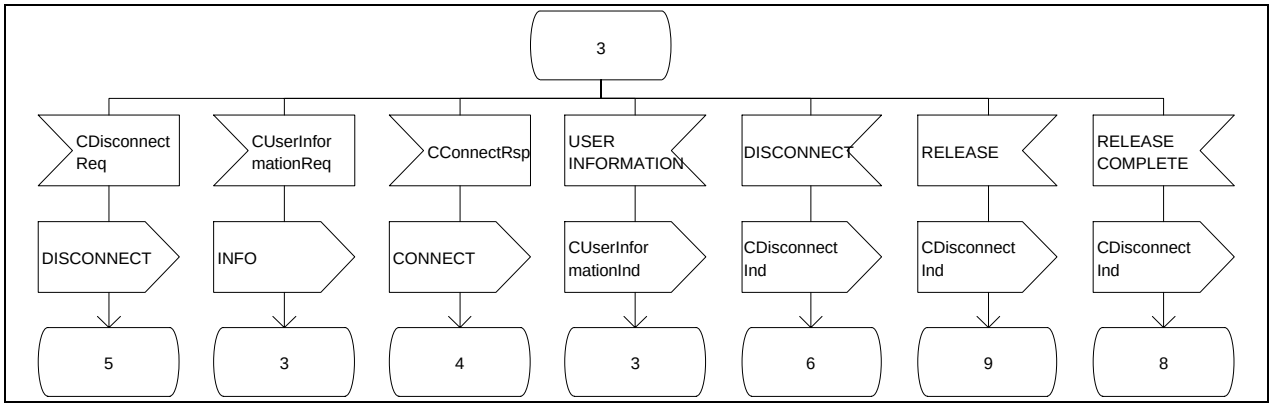


Figure E.4: CALL RECEIVED

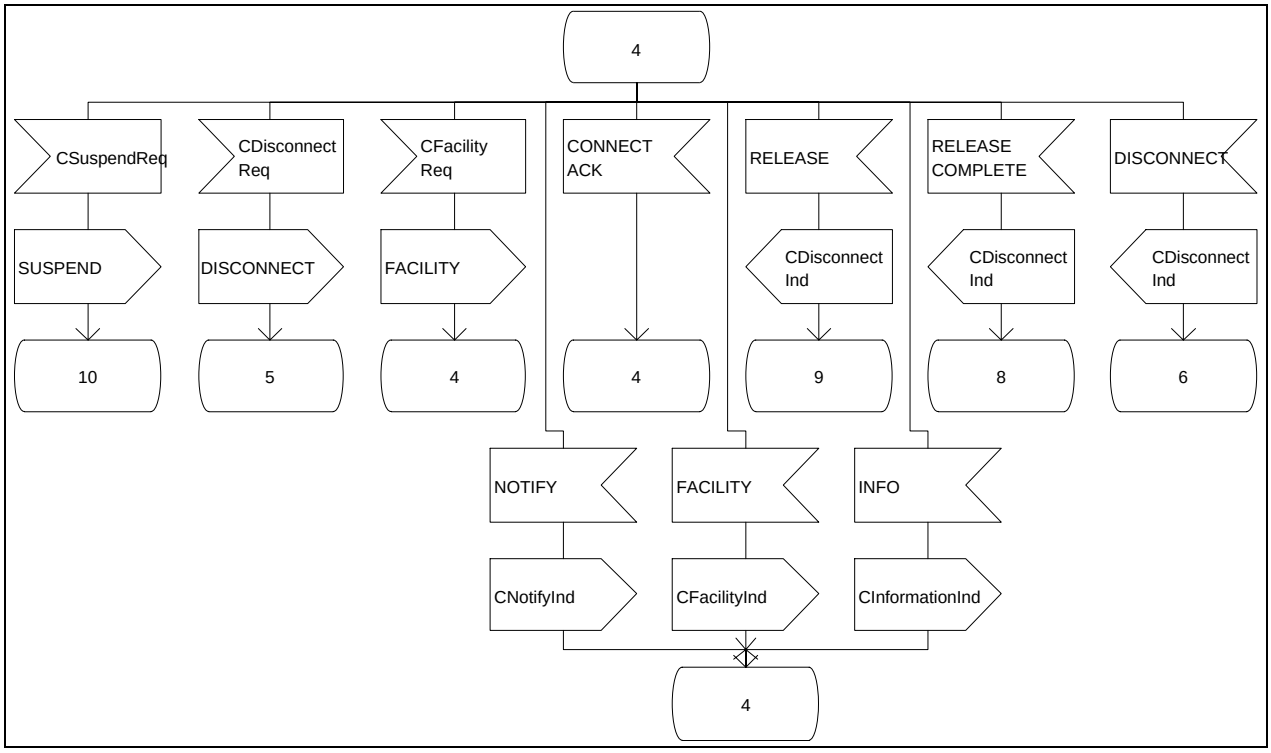


Figure E.5: ACTIVE connection

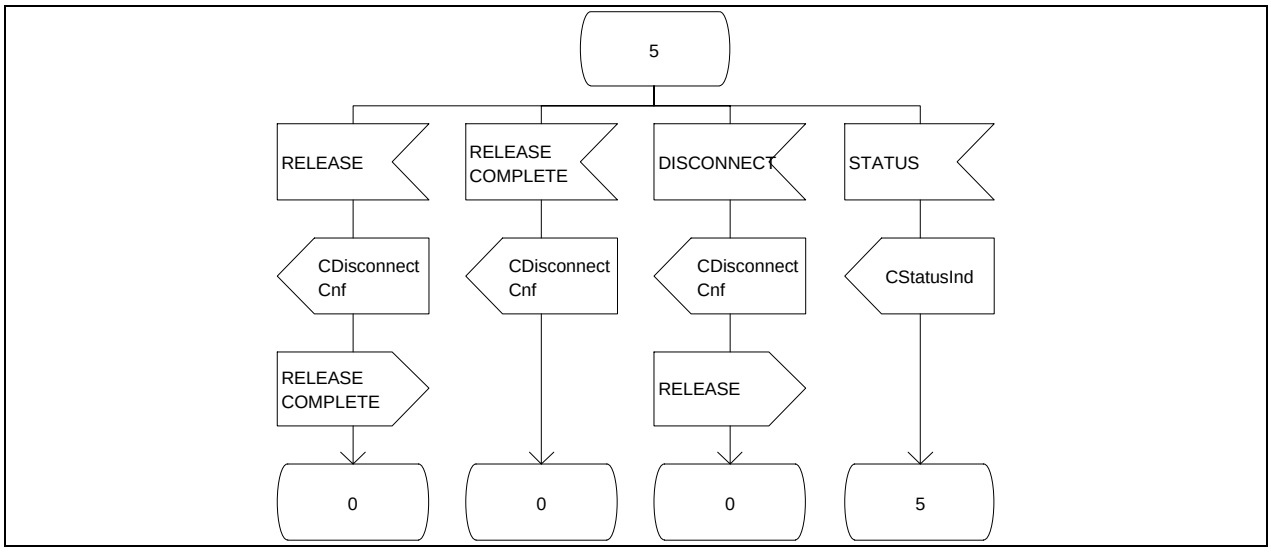


Figure E.6: DISCONNECT request

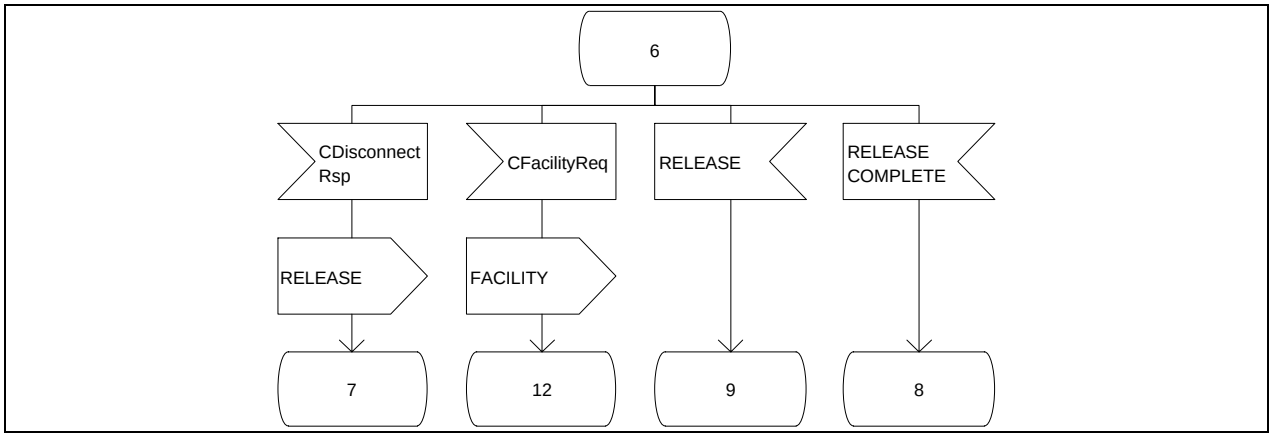


Figure E.7: DISCONNECT indication

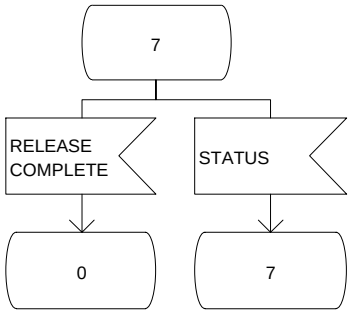


Figure E.8: DISCONNECT pending

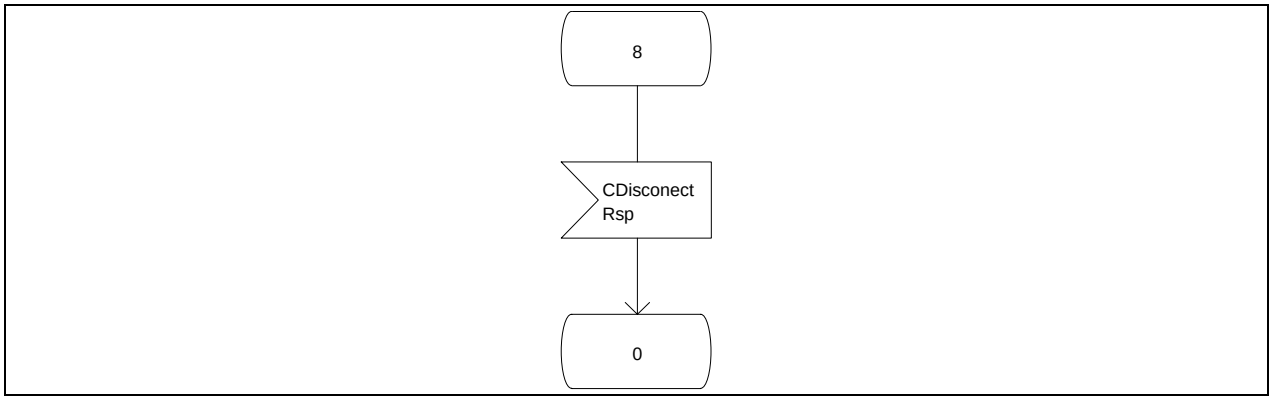


Figure E.9: DISCONNECT response

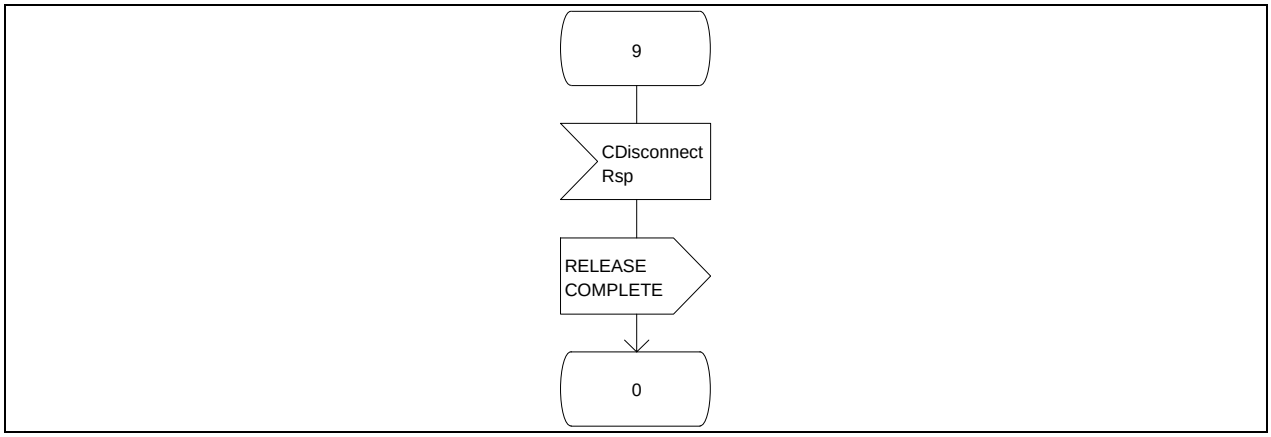


Figure E.10: RELEASE response

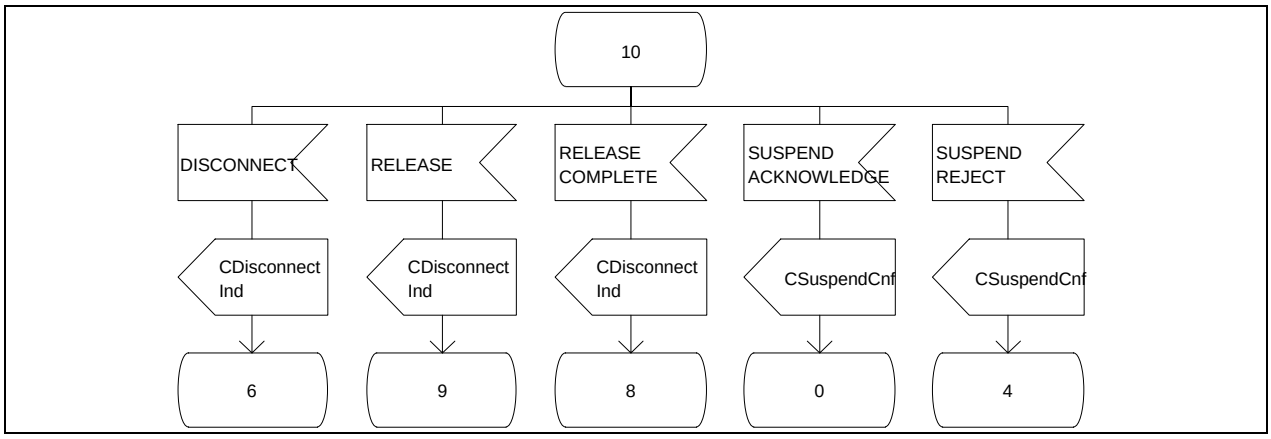


Figure E.11: SUSPEND request

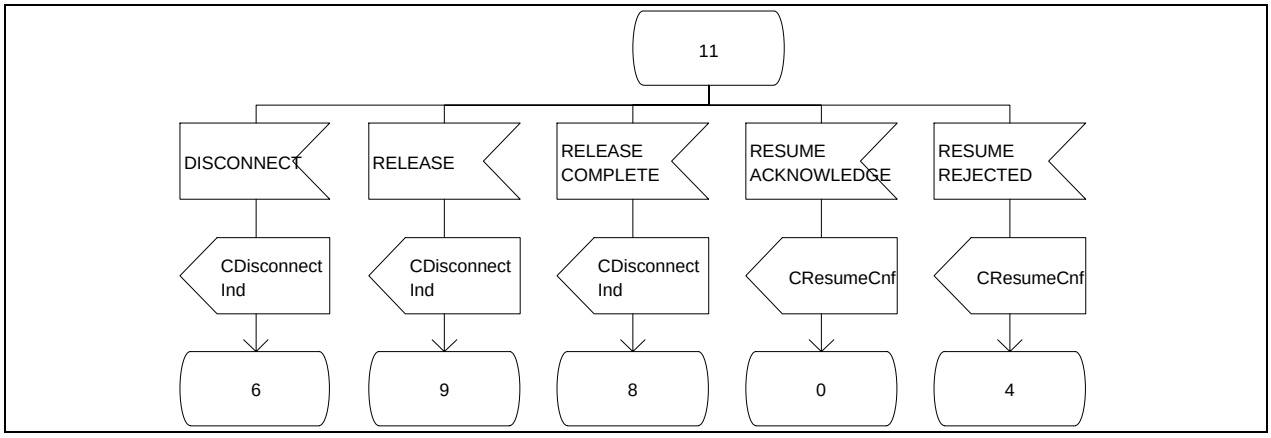


Figure E.12: RESUME request

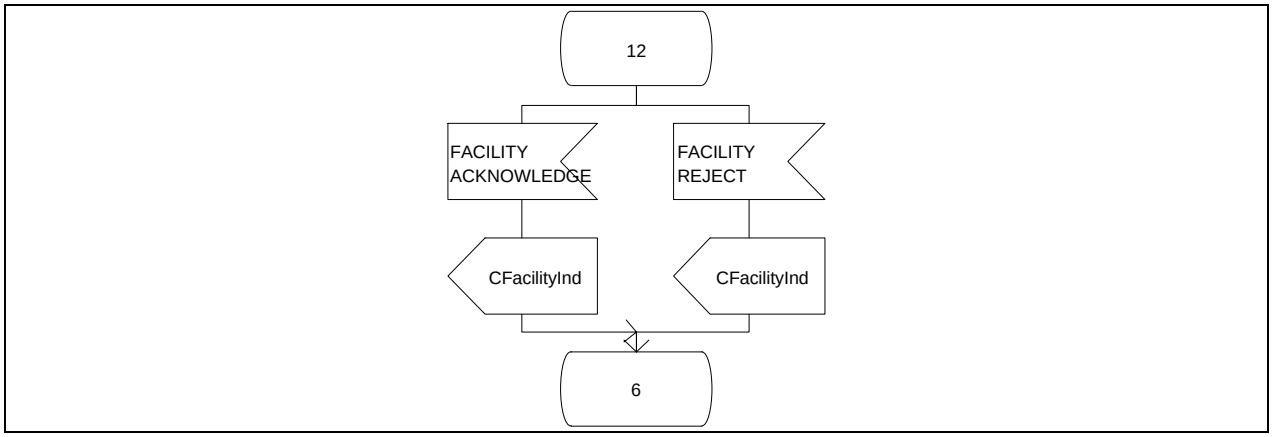


Figure E.13: FACILITY request

E.1.3 Configuration and NAF SDL diagrams for layer one protocols

E.1.3.1 Configuration

E.1.3.1.1 Transparent B-channel access

Table E.1: User Plane configuration for the transparent B-channel access

Parameter	Suggested Default	Comment
Idle flag default value	0xFF	

E.1.4 Configuration and NAF SDL diagrams for layer two protocols

E.1.4.1 Configuration

E.1.4.1.1 ISO 7776 protocol

The following subclauses give the default values parameters to use if they are absent from the parameter list during the NCO creation operation.

Table E.2: User Plane ISO 7776 configuration

Parameter	Suggested Default	Comment
L2FrameSize	128	
L2WindowSize	7	
L2ConnectionMode	Auto	
L2XID		Not used

E.1.4.1.2 PPP protocol

Table E.3: User Plane PPP configuration

Parameter	Suggested Default	Comment
Link Control Protocol Parameters		
- Maximum-Receive-Unit	1 500 (bytes)	Enables a peer to inform the maximum packet size accepted in reception
- Restart timeout	3 (second)	Waiting time for a response to a request packet
- Max terminate	2	Counter for number of terminate requests sent without response
- Max configure	10	Counter for number of configure requests sent without response
- Max failure	10	Counter for number of Configure-Nak received sent before sending a Configure-Reject assuming that the configuration is not converging
- Magic number	None	
- Protocol compression	None	
- Address and Control field Compression	None	
Authentification Protocol		
- type of authentification protocol to be set	None	Enable use of the PPP Authentification Protocols
- Local-ID		Length and name of the local Peer ID
- Local password		Length and name of the local Password
- List of the remote of couple peers "ID/Password"		List of length and name of the remote couple of "ID/Password
-Algorithm		Type of CHAP algorithm used
Line Quality Monitoring - Reporting period	None	Value of maximum time of sending information
FCSAlternatives		Indicate the value of the FCS format to use
SelfDescPadding	None	Indicate the value of the Self Describing Padding to use
CallBack		
- Use of the callback - Number to callback	None	Indicate if the callback option is to be set
CompoundFrame	None	Indicate if the CompoundFrame option is to be set.

E.1.4.1.3 SDLC protocol

Table E.4: User Plane SDLC configuration

Parameter	Suggested Default	Comment
SDLC connection mode	Case 1	SDLC default link station role Case 1 - secondary link station Case 2 - primary link station
SDLC initialisation mode	Case 1	SDLC default initialisation mode Case 1 - send / answer to SNRM Case 2 - send RIM / SIM
SDLC address	0xC1	SDLC default station address
SDLC modulus	8	SDLC default frame numbering modulus
SDLC window size	7	SDLC default window size
SDLC frame size	265	SDLC default maximum frame size excluding the link header and the link trailer.
SDLC timers - T1 - T2 - N2	 2 1 10	SDLC protocol timers Expressed in seconds. Expressed in seconds. Maximum number of unsuccessful retransmission.

E.1.4.1.4 V.110 protocol

Table E.5: User Plane V.110 configuration

Parameter	Suggested Default	Comment
Out of synchronisation timer	3s	Maximum time for resynchronisation
Synchronization timer	10 s	Maximum time for Synchronisation

E.1.4.2 NAF flow diagrams

E.1.4.2.1 ISO 7776 protocol

Table E.6 shows the mapping of User Plane messages to service primitives.

Table E.6: Mapping between User Plane message and protocol messages

Profile A	ISO 7776
UConnectReq	Send SABM(E)
UConnectInd	Received SABM(E)
UConnectRsp	Send UA
UConnectCnf	Received UA
UDisconnectReq	Send DISC
UDisconnectInd	Received DISC or FRMR
UDataReq	Send I frame
UDataInd	Received I frame
UReadyToReceiveReq (busy)	Send RNR
UReadyToReceiveReq (free)	Send RR
UReadyToReceiveInd (busy)	Received RNR
UReadyToReceiveInd (free)	Received RR
NOTE: REJ frames and frames numbering are handled transparently by the NAF.	

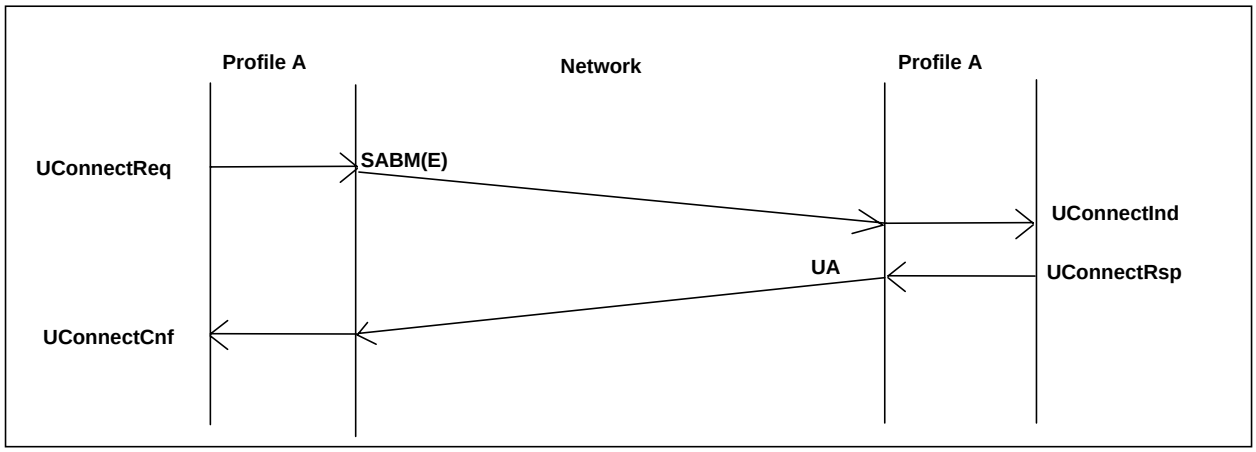


Figure E.14: CONNECTION PHASE

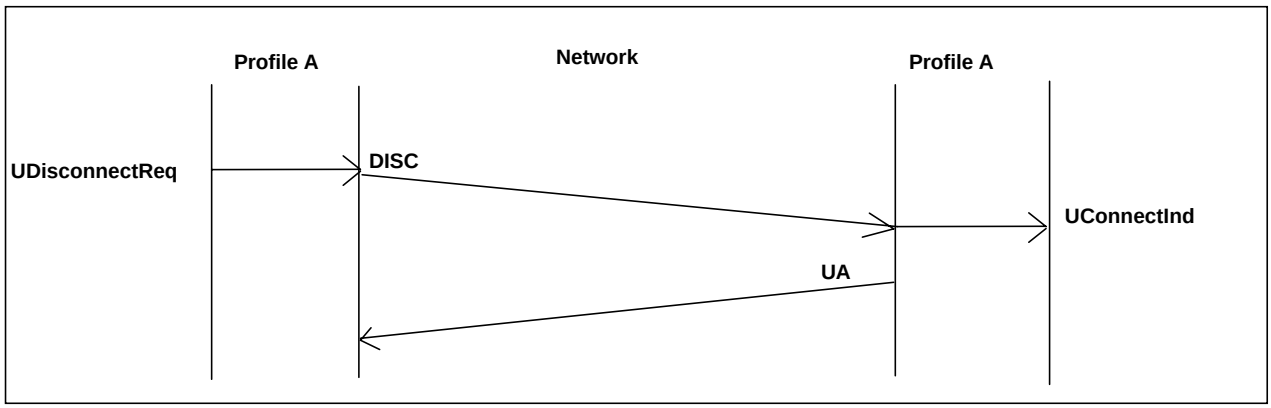


Figure E.15: DISCONNECTION PHASE

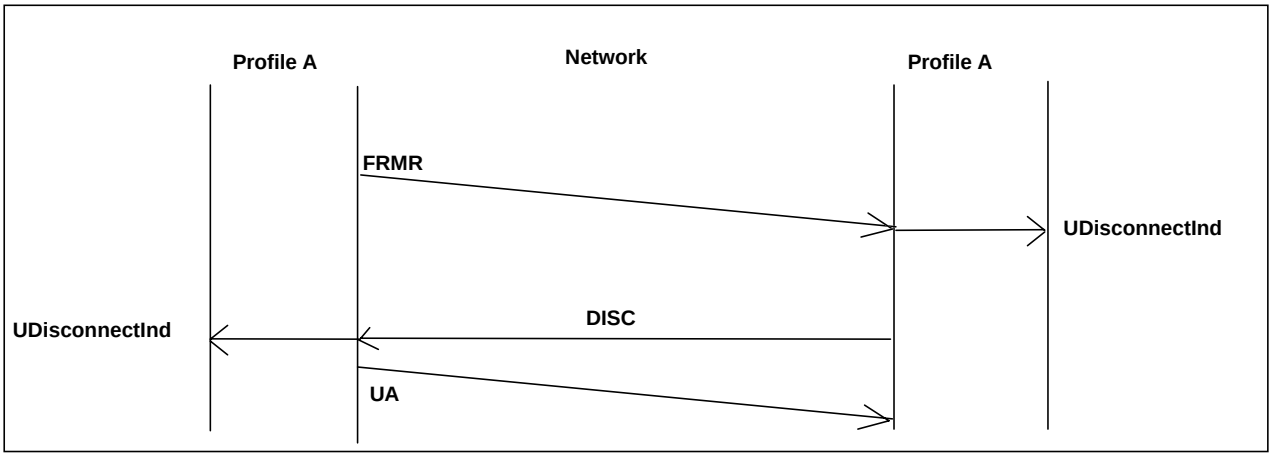


Figure E.16: ERROR SITUATION

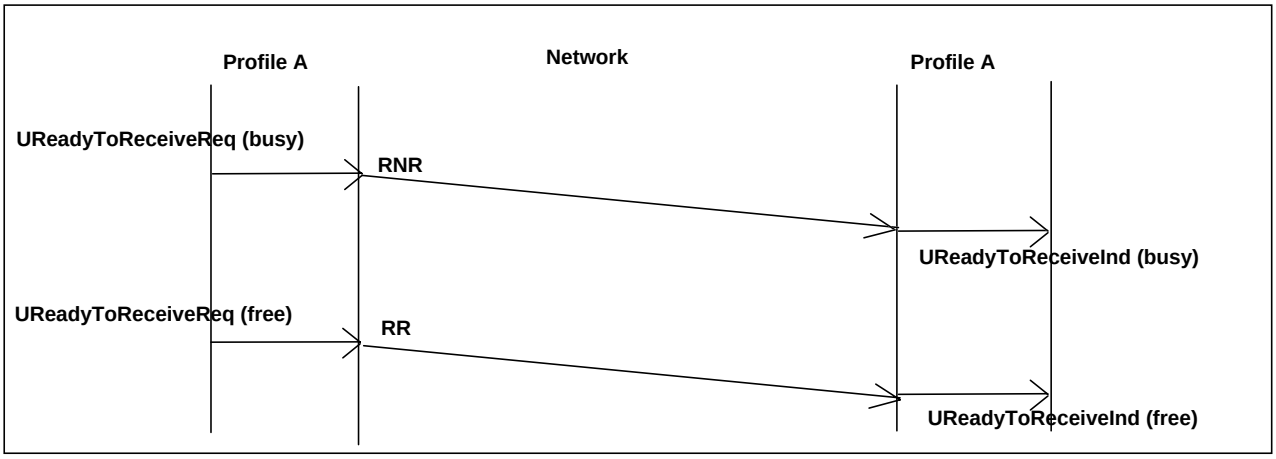


Figure E.17: FLOW CONTROL SITUATION

E.1.4.2.2 HDLC protocol

Table E.7 shows the mapping of User Plane messages to service primitives.

Table E.7: Mapping between User Plane message and protocol messages

PCI Message	Primitive
UDataReq	I
UDataInd	I

E.1.4.2.3 HDLC protocol with error

Table E.8 shows the mapping of User Plane messages to service primitives.

Table E.8: Mapping between User Plane message and protocol messages

PCI Message	Primitive
UDataReq	I
UDataInd	I

E.1.4.2.4 PPP protocol

Table E.9 shows the mapping of User Plane messages to service primitives.

Table E.9: Mapping between User Plane message and protocol messages

PCI Message	Primitive
UConnectReq	UI - CONFIGURE REQUEST
UConnectInd	UI - CONFIGURE REQUEST
UConnectRsp	UI - CONFIGURE ACK / NACK
UConnectCnf	UI - CONFIGURE ACK / NACK
UDisconnectReq	UI - TERMINATE REQUEST
UDisconnectInd	UI - CONFIGURE REJECT / TERMINATE REQUEST
UDataReq	UI - INFO (NCP)
UDataInd	UI - INFO (NCP)
UErrorInd	UI - PROTOCOL REJECT CODE REJECT

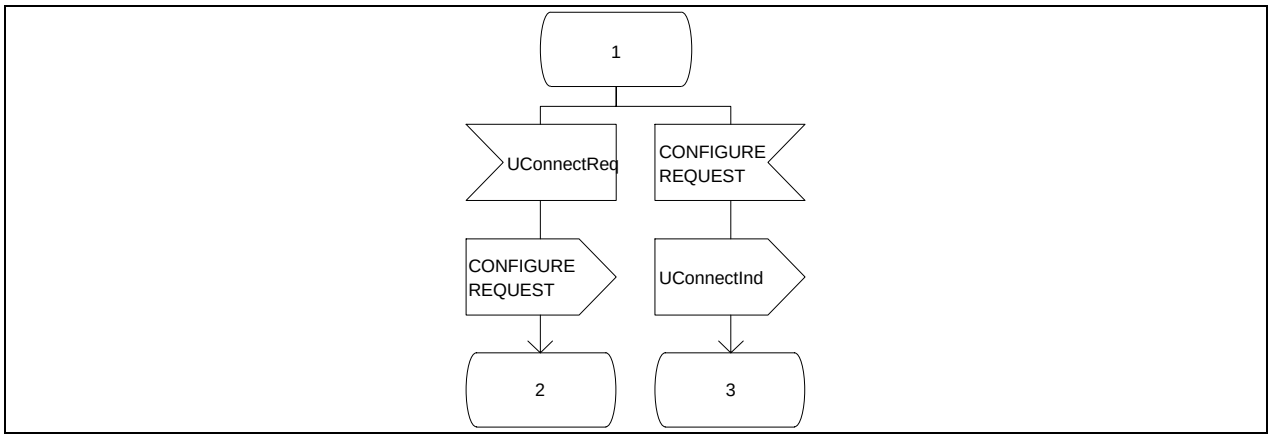


Figure E.18: IDLE

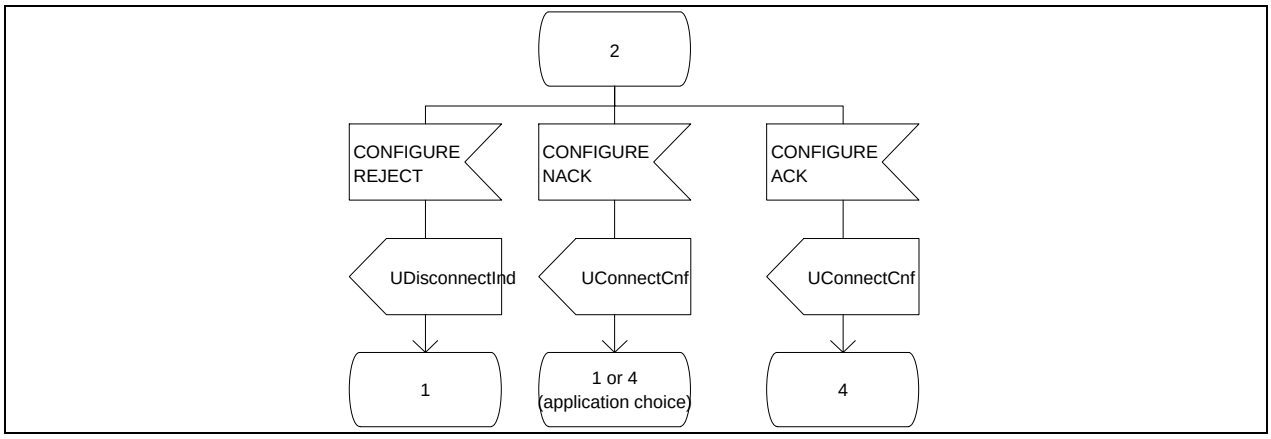


Figure E.19: OUTGOING CONNECTION PENDING

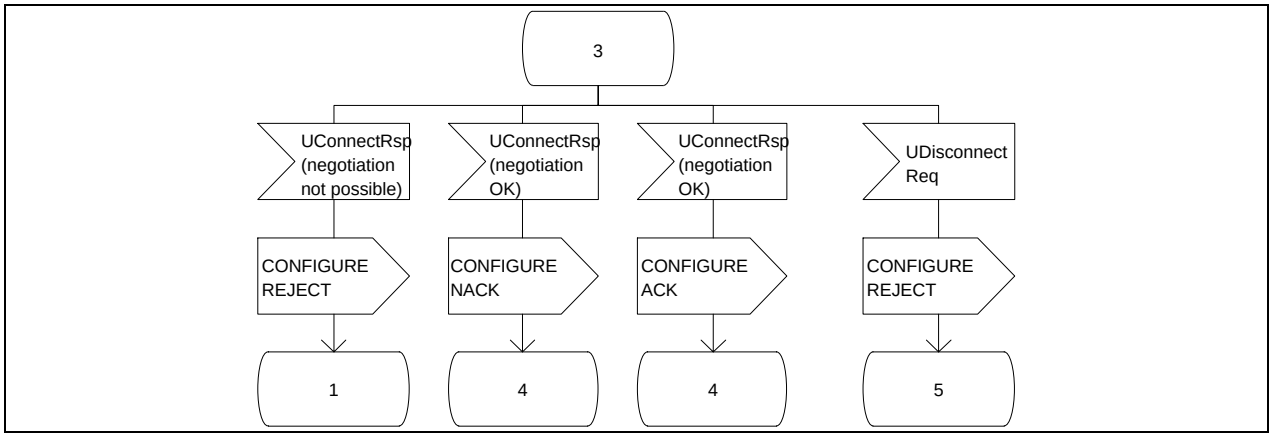


Figure E.20: INCOMING CONNECTION PENDING

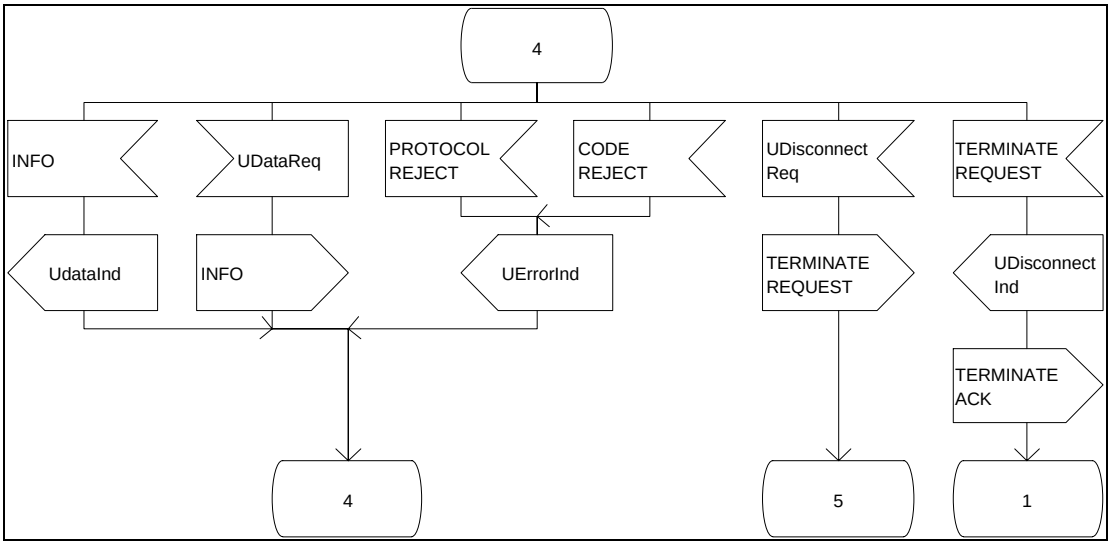


Figure E.21: DATA TRANSFER READY

E.1.4.2.5 SDLC protocol

Table E.10 shows the mapping of User Plane messages to service primitives.

Table E.10: Mapping between User Plane message and protocol messages

PCI Message	Primitive
UConnectReq	XID P
UConnectInd	XID P
UConnectRsp	XID F
UConnectCnf	XID F
UDisconnectReq	DISC P
UDisconnectInd	RD F ou DM F
UDataReq	I
UDataInd	I
UExpeditedDataReq	UI
UExpeditedDataInd	UI
UReadyToReceiveReq	Local meaning
UReadyToReceiveInd	Local meaning

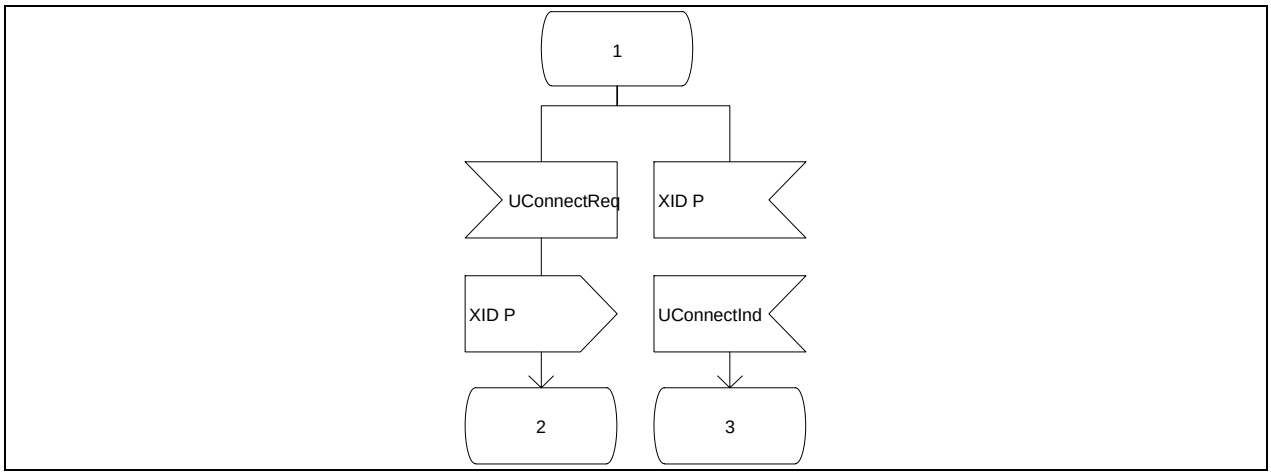


Figure E.22: IDLE

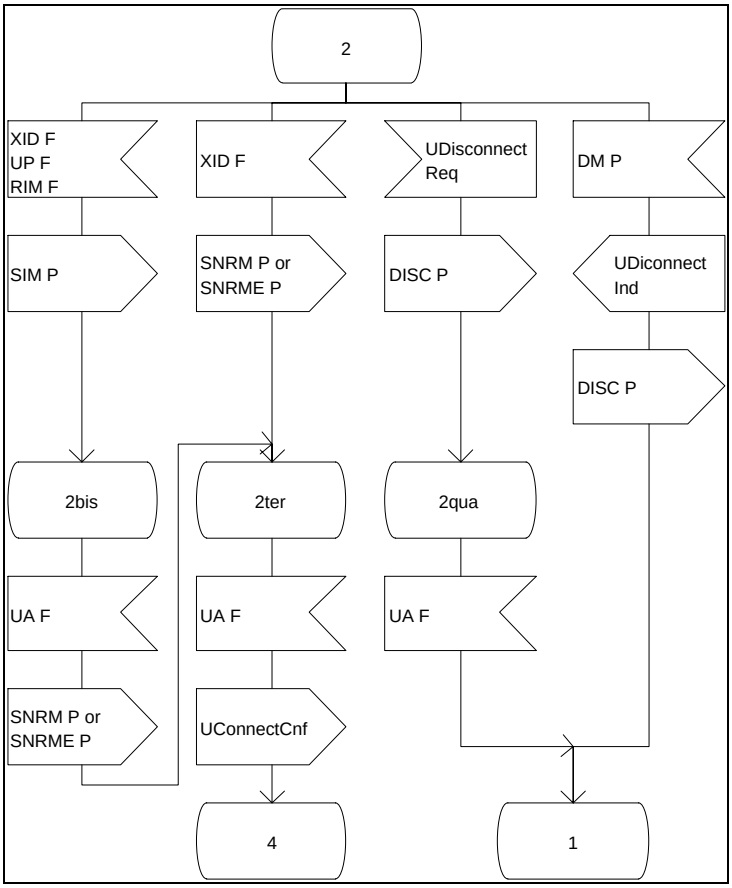


Figure E.23: OUTGOING CONNECTION PENDING (primary)

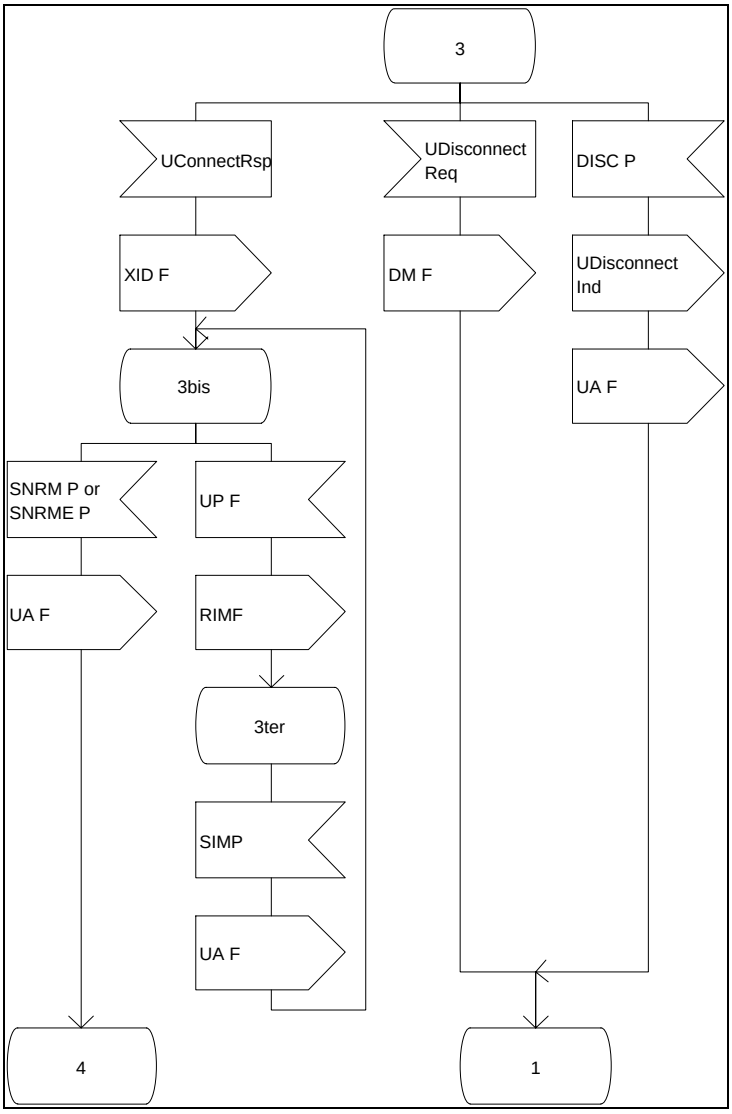


Figure E.24: INCOMING CONNECTION PENDING (secondary)

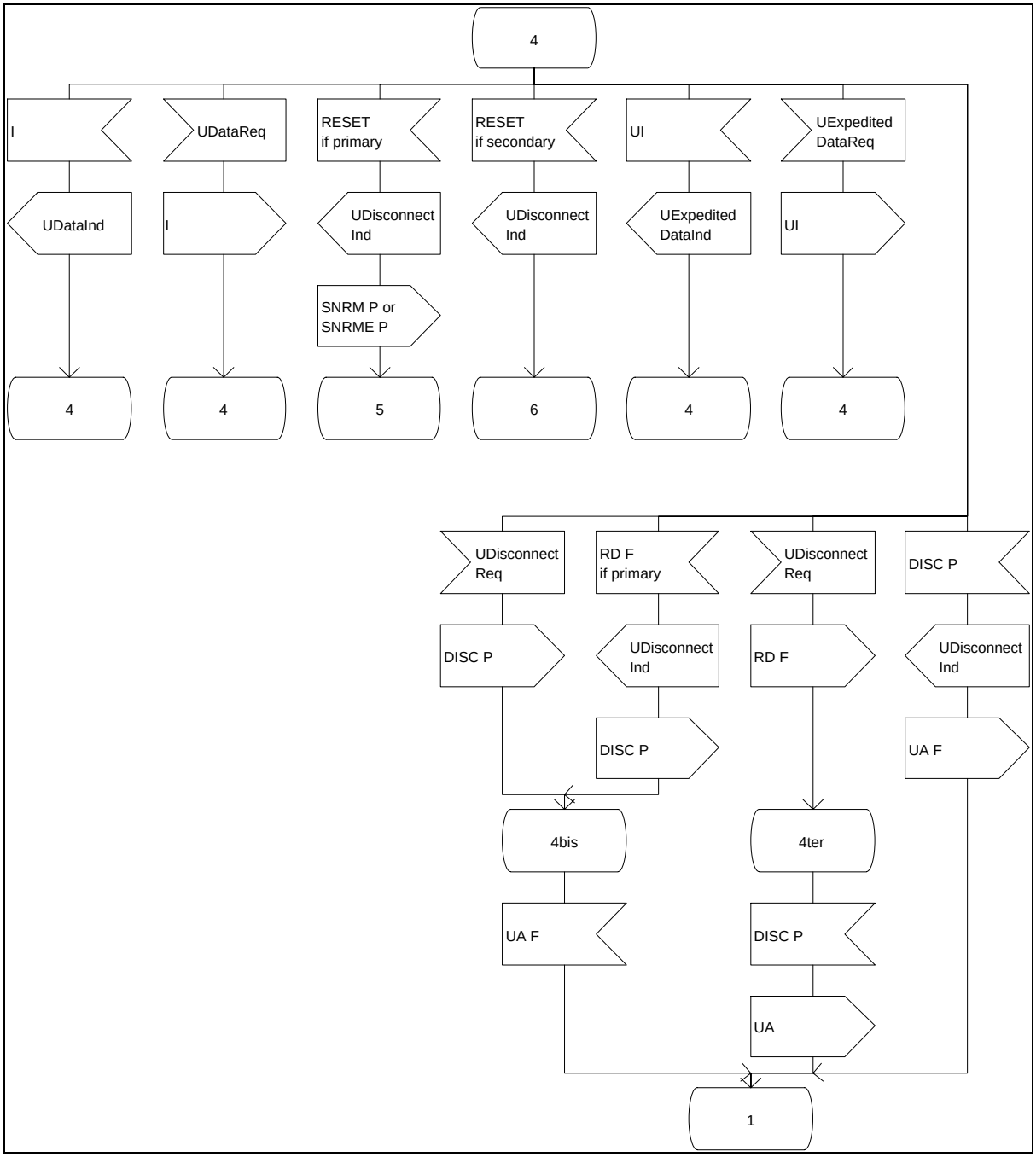


Figure E.25: DATA TRANSFER READY

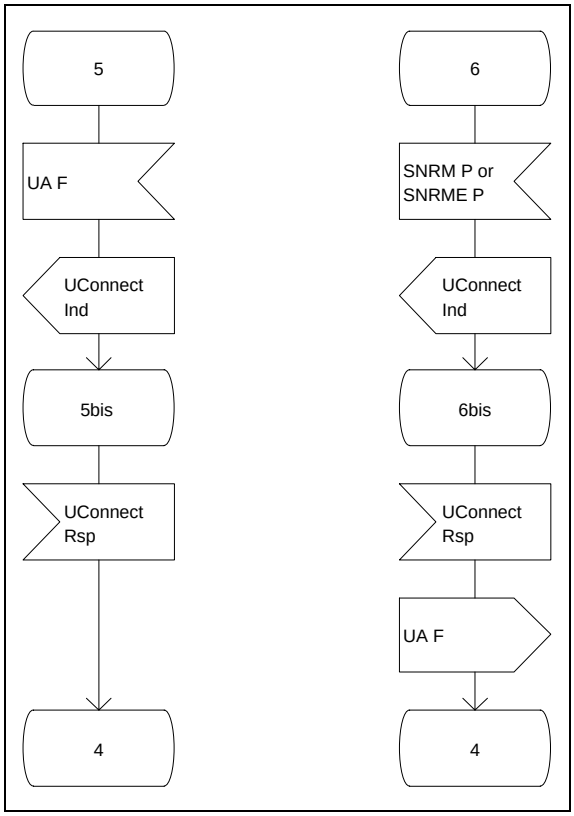


Figure E.26: RESET

E.1.4.2.6 V.110 protocol

Table E.11 shows the mapping of User Plane messages to service elements. For the V.110 protocol there is no direct link between user connection messages and protocol frames. The connection phase consists of synchronisation and negotiation. It begin without application demand, when the ISDN channel is established.

To make the PCI messages meaning easier to understand, table E.11 show a theoretical mapping between User Plane messages and V.110 frames.

Table E.11: Mapping between User Plane message and V.110 frame

PCI Message	Frame
UConnectReq	Frame "Synchronisation" (bit S = bit X = OFF) Local meaning.
UConnectInd	Frame "Ready" (bit S = bit X = ON) - Local meaning: Remote synchronisation has been received.
UConnectRsp	Frame "Synchronisation" (bit S = bit X = OFF) - Local meaning. (note)
UConnectCnf	Frame "Ready" (bit S = bit X = ON) (a negotiation delay may be necessary before data transfer is ready)
UDisconnectReq	Frame with bit S = OFF, bit X = ON, D=0
UDisconnectInd	Frame with bit S = OFF, bit X = ON, D=0
UDataReq	Data frame
UDataInd	Data frame
UReadyToReceiveReq	Local meaning
UReadyToReceiveInd	Local meaning
NOTE: Sending a UConnectRsp does not mean negotiation is finished. Data transfer can be unavailable for a time. In this case the PUF is sent a NAFBusy error (see figure E.28).	

The following figures (E.27 to E.31) show more general cases, but not every possible situation.

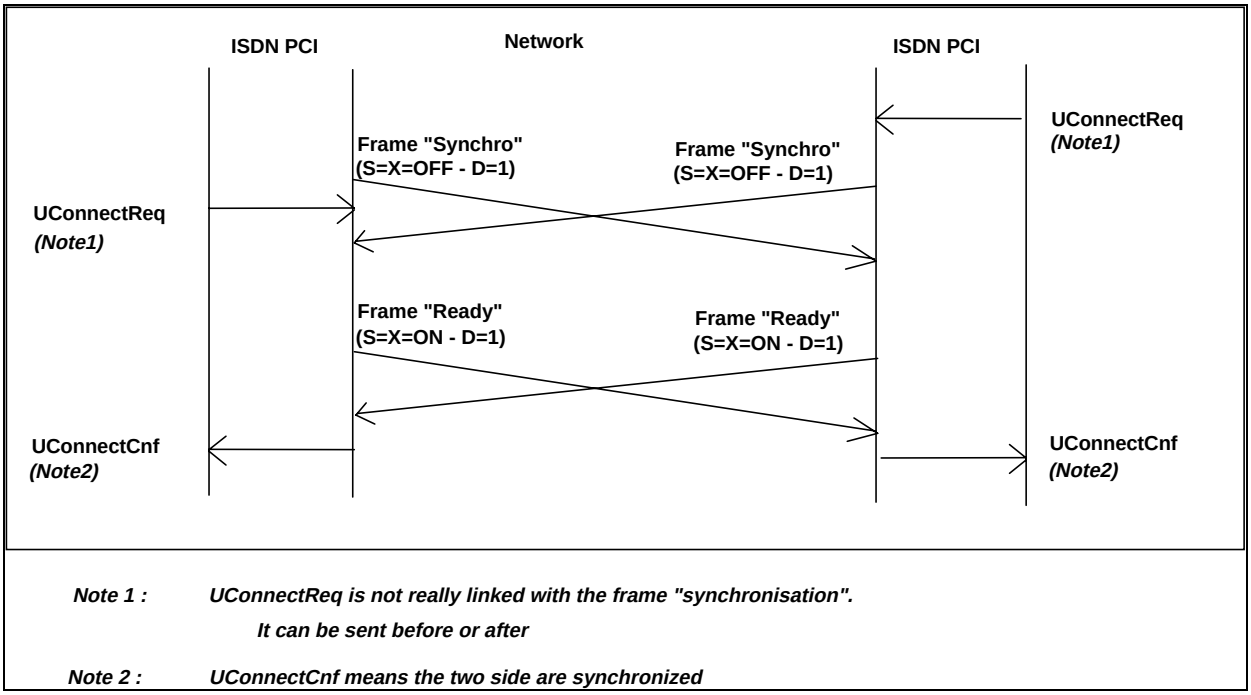
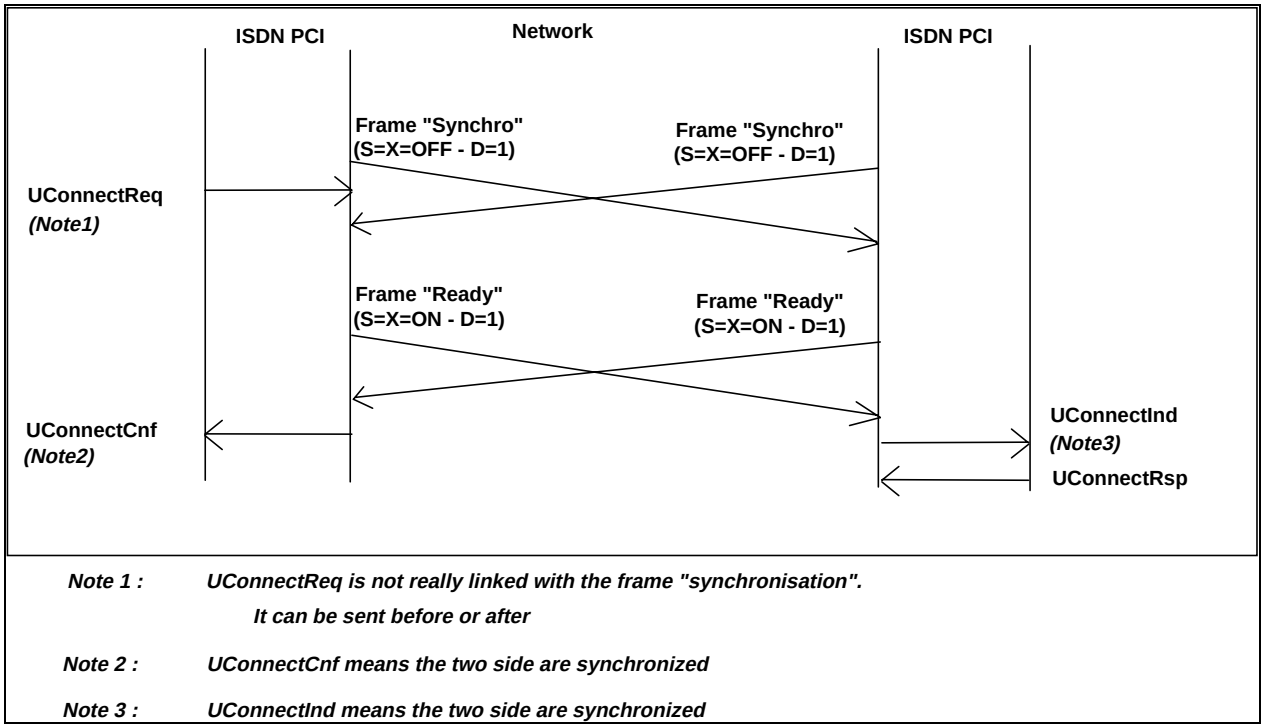


Figure E.26: CONNECTION PHASE



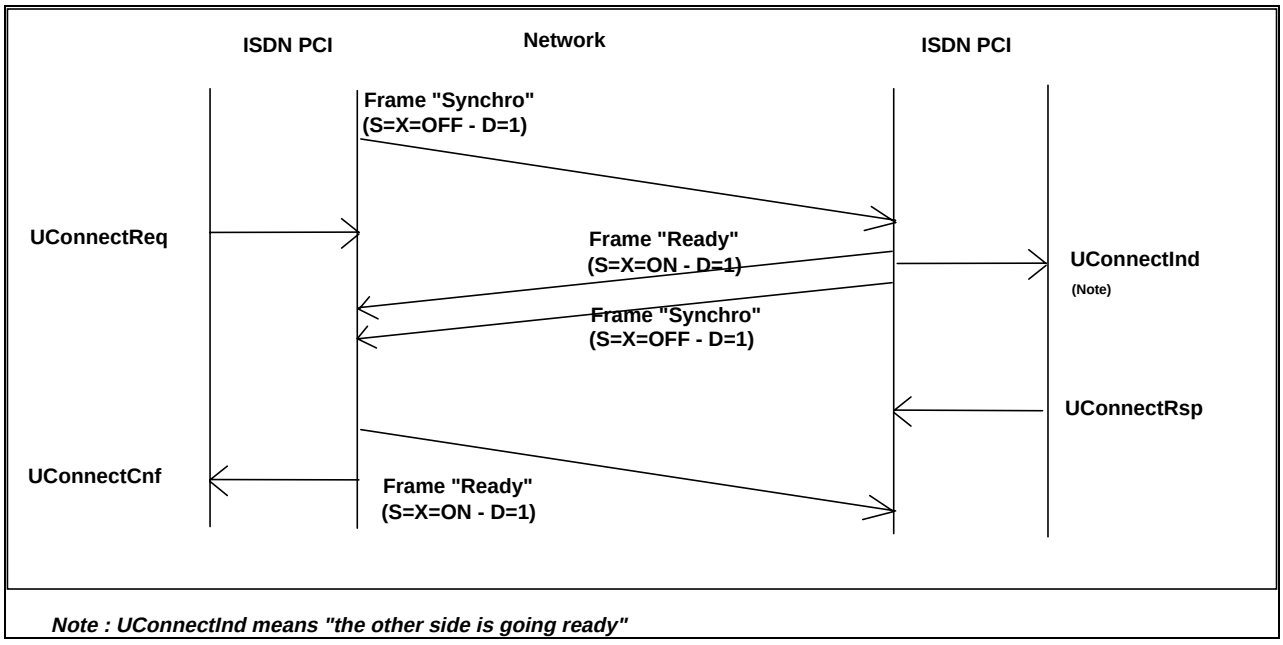
Note 1 : UConnectReq is not really linked with the frame "synchronisation". It can be sent before or after

Note 2 : UConnectCnf means the two side are synchronized

Note 3 : UConnectInd means the two side are synchronized

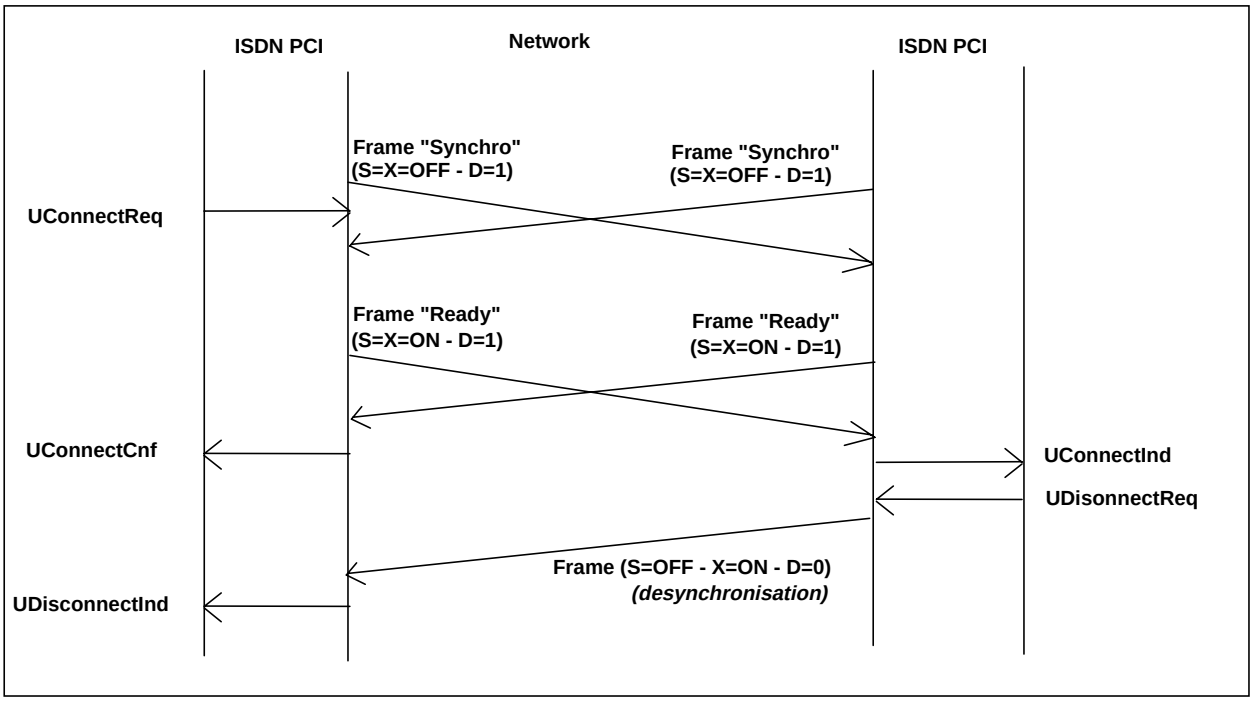
Figure E.27: CONNECTION PHASE

The following figure shows another possible situation. It is a theoretical situation: the low layer V.110 module generally begins synchronisation phase just when the B-channel is established.



Note : UConnectInd means "the other side is going ready"

Figure E.28: CONNECTION PHASE



NOTE: Depending on the V.110BChannelDisconnecion parameter, a user disconnection may imply the B-channel disconnection

Figure E.29: REMOTE DISCONNECTION DURING CONNECTION PHASE

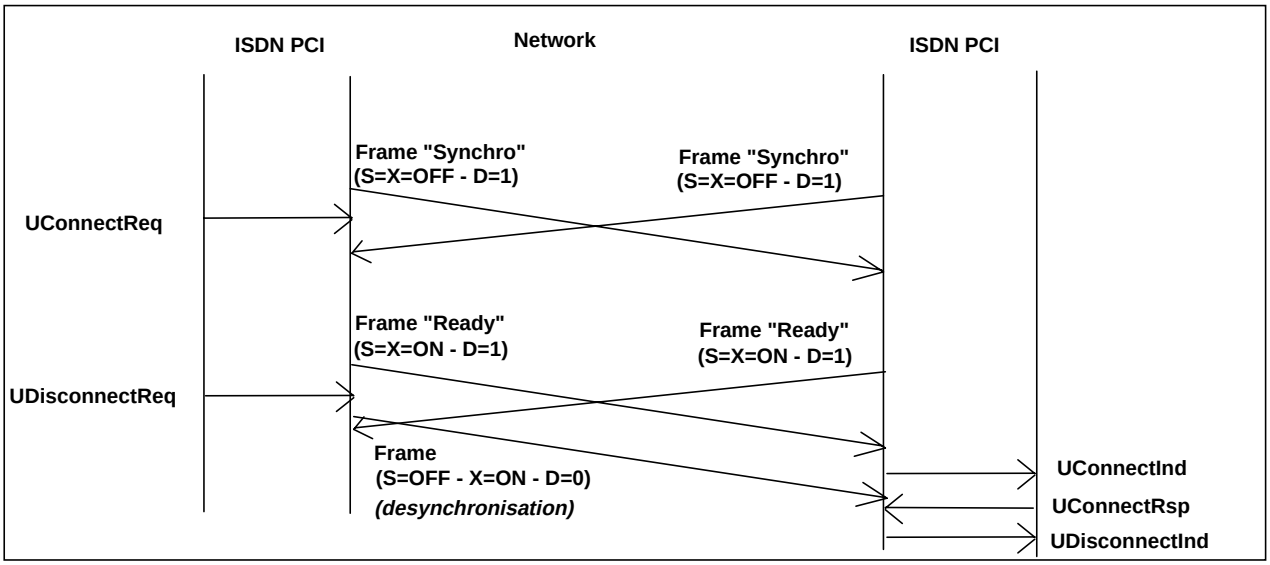


Figure E.30: DISCONNECTION DURING CONNECTION PHASE

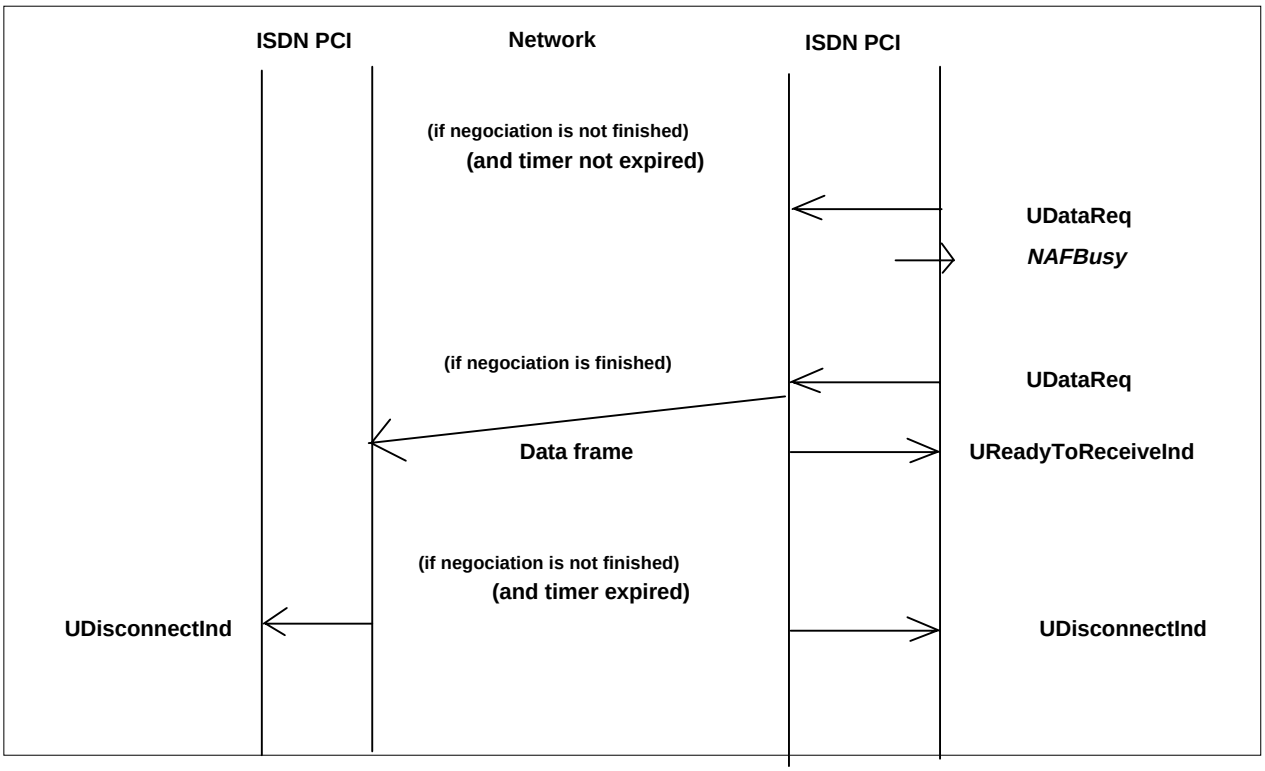


Figure E.31: DATA TRANSFER

E.1.5 Configuration and NAF SDL Diagrams for layer three protocols

E.1.5.1 Configuration

E.1.5.1.1 T.90 protocol

Table E.12: User Plane T.90 configuration

Parameter	Suggested Default	Comment
X.25 Network Type	0	Allows NAF to adapt for different country implementations of X.25.
X.25 Recommendation	CCITT88	Level of X.25 Recommendation supported.
Layer 3 sequence numbering	8	
Layer 3 Maximum Window Size	7	
Layer 3 Default Window Size	3	
Layer 3 Maximum Packet Size	4096	
Layer 3 Default Packet Size	128	
Layer 3 Default Connection Mode	Auto	Auto - Act as DTE when calling, act as DCE when called. DXE - use Restart Packet to determine DTE or DCE role as in ISO 8208 [3]. DTE - Act as DTE. DCE - Act as DCE.
Lowest number of Incoming SVC (LIC)	1	
Highest number of incoming SVC (HIC)	1	
Lowest number of Two way SVC (LTC)	0	
Highest number of Two way SVC (HTC)	0	
Lowest number of outgoing SVC (LOC)	0	
Highest number of outgoing SVC (HOC)	0	
Layer 3 Timers		NAF may wish to provide PUF user the ability to configure timers.
Layer 2 Default Connection Mode	Auto	Auto - When calling act as DTE, when called act as DCE. DTE as defined in ISO 7776 [4]. DCE as defined in ISO 7776 [4].
Layer 2 B-channel modulus	8	Shall be 128 for X.25 on D-channel.
Layer 2 Window Size	7	
Layer 2 Frame Size	128	
Layer 2 activation type	Case 1	Case1 - send SABM/SABME when calling, do not send when called. Case2 - send SABM/SABME when called, do not send when calling. Passive - do not send SABM/SABME when initiated. Active - send SABM/SABME when initiated.
Layer 2 Timers - T1 - T1 - N2	 5 1 5	 Expressed in seconds. Expressed in seconds. Maximum number of unsuccessful retransmission.

E.1.5.1.2 ISO 8208 protocol

Table E.13: User Plane ISO 8208 configuration

Parameter	Suggested Default	Comment
X.25 Network Type	0	Allows NAF to adapt for different country implementations of X.25.
X.25 Recommendation	CCITT 1988	Level of X.25 Recommendation supported.
Layer 3 sequence numbering	8	
Layer 3 Maximum Window Size	7	
Layer 3 Default Window Size	3	
Layer 3 Maximum Packet Size	4096	
Layer 3 Default Packet Size	128	
Layer 3 Default Connection Mode	Auto	Auto - Act as DTE when calling, act as DCE when called. DXE - use Restart Packet to determine DTE or DCE role as in ISO 8208 [3]. DTE - Act as DTE. DCE - Act as DCE.
Lowest number of Incoming SVC (LIC)	1	
Highest number of incoming SVC (HIC)	1	
Lowest number of Two way SVC (LTC)	0	
Highest number of Two way SVC (HTC)	0	
Lowest number of outgoing SVC (LOC)	0	
Highest number of outgoing SVC (HOC)	0	
Layer 3 Timers		NAF may wish to provide PUF user the ability to configure timers.
Layer 2 Default Connection Mode	Auto	Auto - When calling act as DTE, when called act as DCE. DTE as defined in ISO 7776. DCE as defined in ISO 7776.
Layer 2 B-channel modulus	8	note shall be 128 for X.25 on D-channel.
Layer 2 Window Size	7	
Layer 2 Frame Size	128	
Layer 2 activation type	Case 1	Case1 - send SABM/SABME when calling, do not send when called. Case2 - send SABM/SABME when called, do not send when calling. Passive - do not send SABM/SABME when initiated. Active - send SABM/SABME when initiated.
Layer 2 Timers - T1 - T2 - N2	 5 1 5	 Expressed in seconds. Expressed in seconds. Maximum number of unsuccessful retransmission.

E.1.5.1.3 T.70 protocol

Table E.14: User Plane T.70 configuration

Parameter	Suggested Default	Comment
Layer 3 Maximum Packet Size	2 048	
Layer 3 Default Packet Size	128	
Layer 2 Timers - T1 - T1 - N2	 5 1 5	 Expressed in seconds. Expressed in seconds. Maximum number of unsuccessful retransmission.

E.1.5.2 NAF SDL diagrams

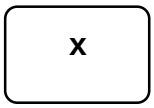
The mapping of User Plane messages to protocol messages depends on whether the NAF is providing the co-ordination function for a particular Control Plane connection.

When the NAF is providing the co-ordination function the mapping of X.213 service primitives [6] to ETS 300 102 [2] messages and X.25 packets is explained in ISO/IEC 9574 [9] and ISO/IEC 8878.

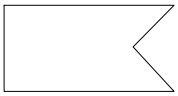
When the NAF is not providing the co-ordination function the mapping of X.213 service primitives to X.25 packets is explained in ISO/IEC 8878.

Some SDL diagrams are given to explain the relation between user messages and network primitives. These diagrams do not cover every case. They only present some of the possible cases.

The following symbols are used within this description. A full description of the symbols and their meaning is given in ITU-T Recommendation Z.100.



State Symbol

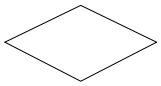


Input (from Network)

Input (from PUF)

Output (to Network)

Output (to PUF)



Decision Symbol

E.1.5.2.1 T.90 protocol

Table E.15 shows the mapping of User Plane messages to X.213 service primitives.

Table E.15: Mapping between User Plane message and protocol messages

PCI Message	X.213 Primitive
UConnectReq	N-CONNECT request
UConnectInd	N-CONNECT indication
UConnectRsp	N-CONNECT response
UConnectCnf	N-CONNECT confirm
UDisconnectReq	N-DISCONNECT request
UDisconnectInd	N-DISCONNECT indication
UDataReq	N-DATA request
UDataInd	N-DATA indication
UResetReq	N-RESET request
UResetInd	N-RESET indication
UResetRsp	N-RESET response
UResetCnf	N-RESET confirm
UReadyToReceiveReq	Not equivalent to an X.213 primitive
UReadyToReceiveInd	Not equivalent to an X.213 primitive

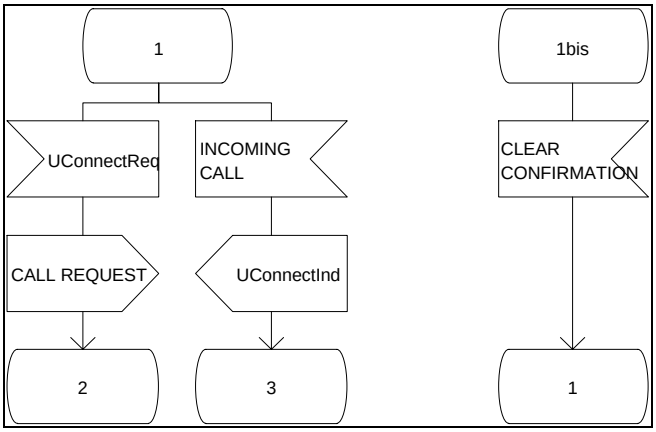


Figure E.32: IDLE

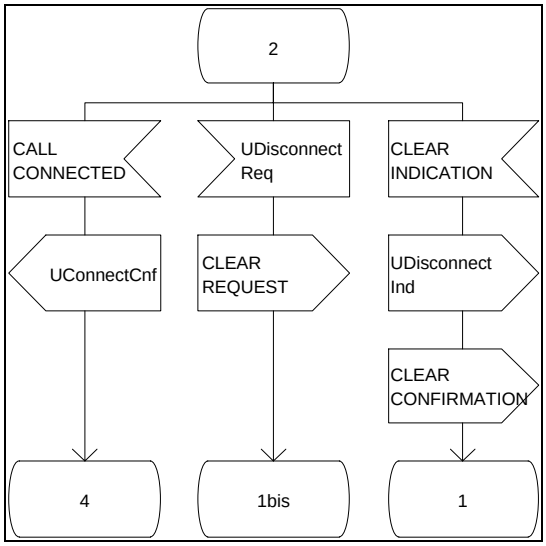


Figure E.33: OUTGOING CONNECTION PENDING

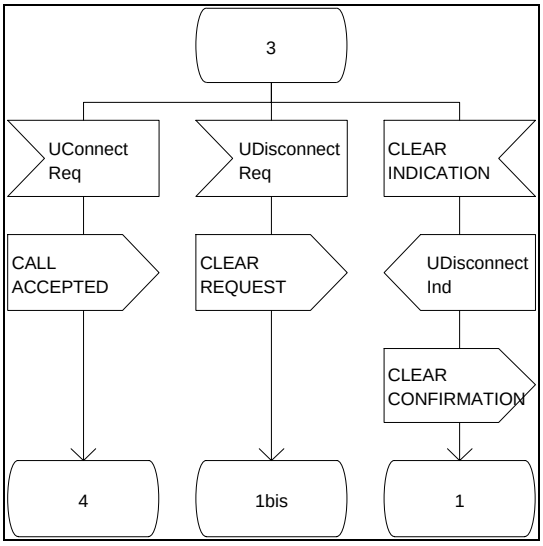


Figure E.34: INCOMING CONNECTION PENDING

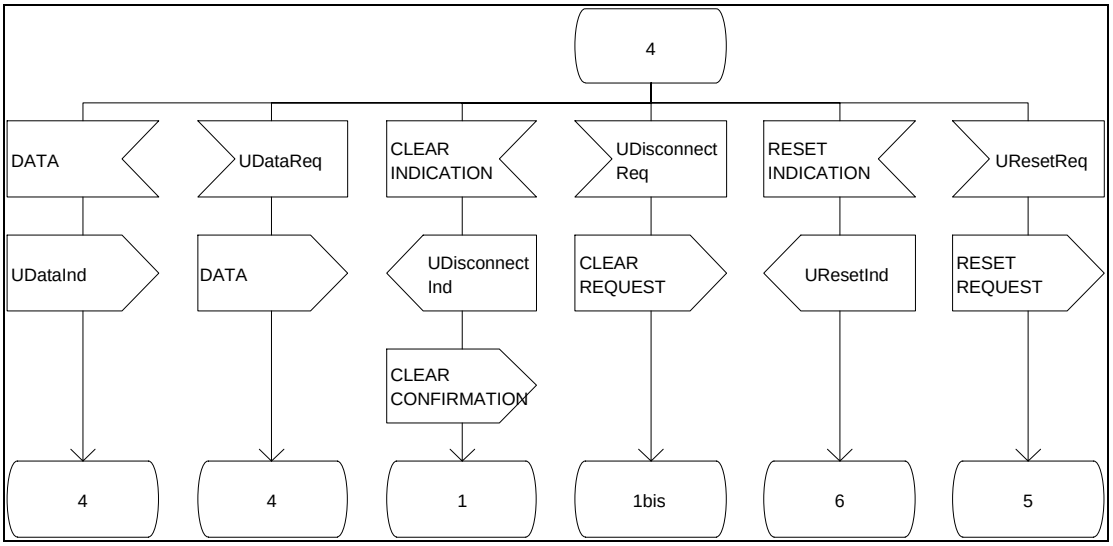


Figure E.35: DATA TRANSFER READY

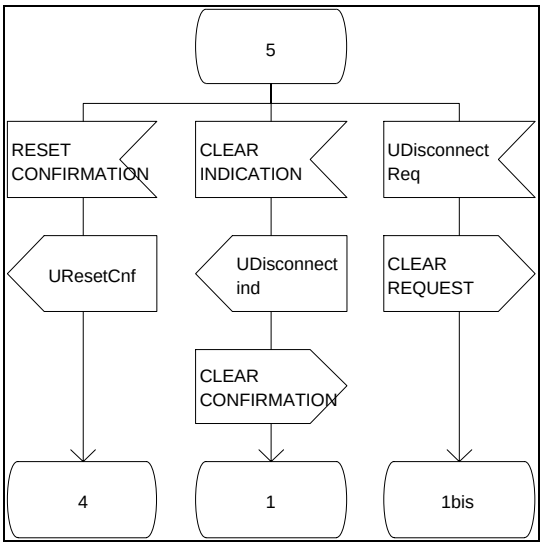


Figure E.36: PUF INVOKED RESET PENDING

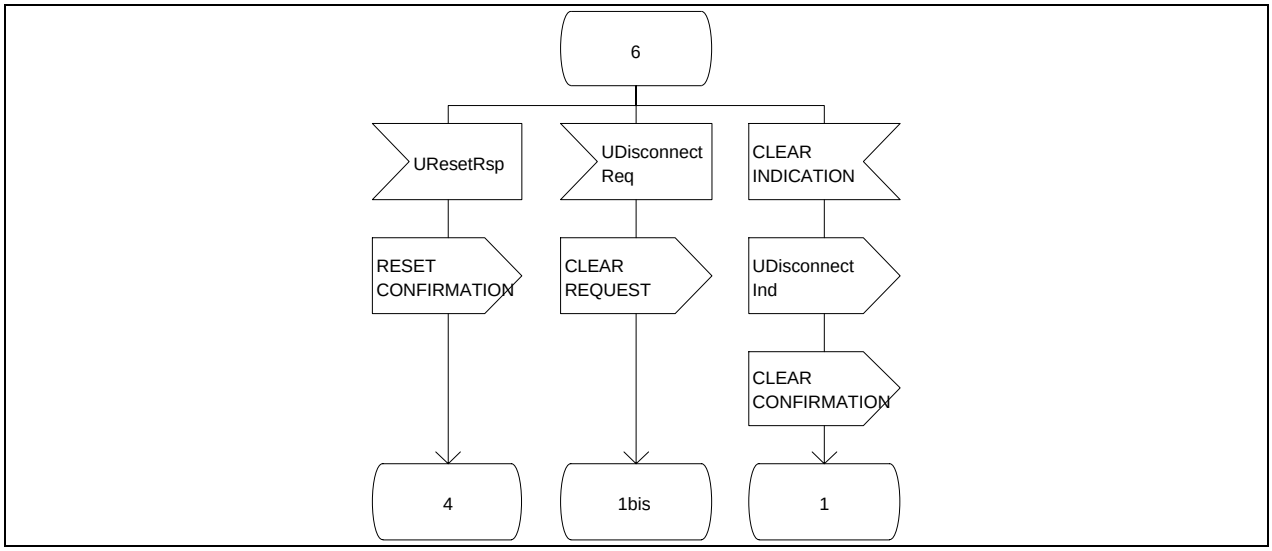


Figure E.37: NETWORK INVOKED RESET PENDING

E.1.5.2.2 ISO 8208 protocol

Table E.16 shows the mapping of User Plane messages to X.213 service primitives.

Table E.16: Mapping between User Plane message and protocol messages

PCI Message	X.213 Primitive
UConnectReq	N-CONNECT request
UConnectInd	N-CONNECT indication
UConnectRsp	N-CONNECT response
UConnectCnf	N-CONNECT confirm
UDisconnectReq	N-DISCONNECT request
UDisconnectInd	N-DISCONNECT indication
UDataReq	N-DATA request
UDataInd	N-DATA indication
UExpeditedDataReq	N-EXPEDITED-DATA request
UExpeditedDataInd	N-EXPEDITED-DATA indication
UResetReq	N-RESET request
UResetInd	N-RESET indication
UResetRsp	N-RESET response
UResetCnf	N-RESET confirm
UDataAcknowledgeReq	N-DATA-ACKNOWLEDGE request
UDataAcknowledgeInd	N-DATA-ACKNOWLEDGE indication
UReadyToReceiveReq	Not equivalent to an X.213 primitive
UReadyToReceiveInd	Not equivalent to an X.213 primitive

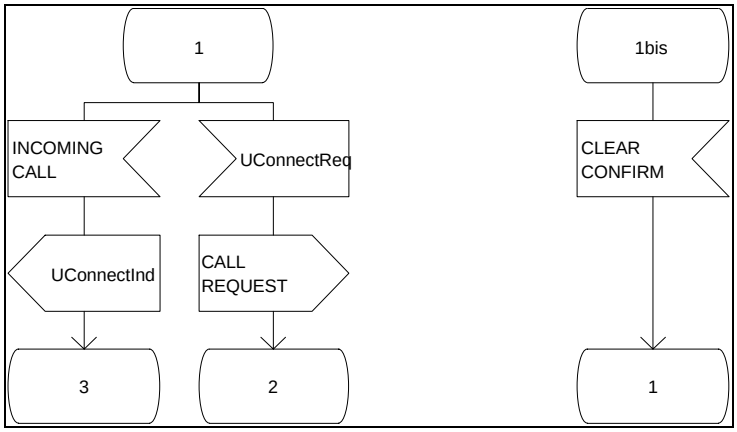


Figure E.38: IDLE

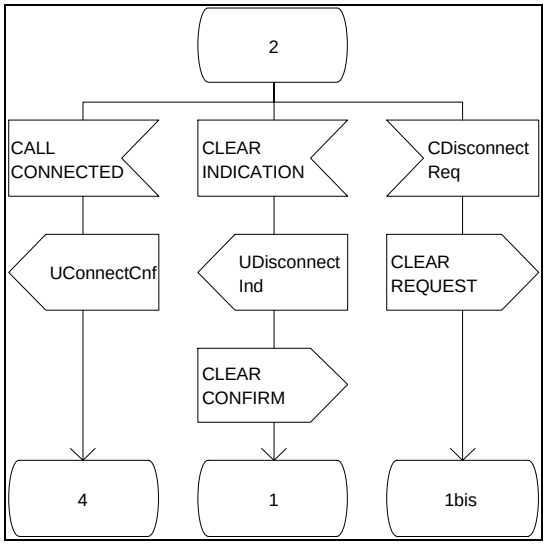


Figure E.39: OUTGOING CONNECTION PENDING

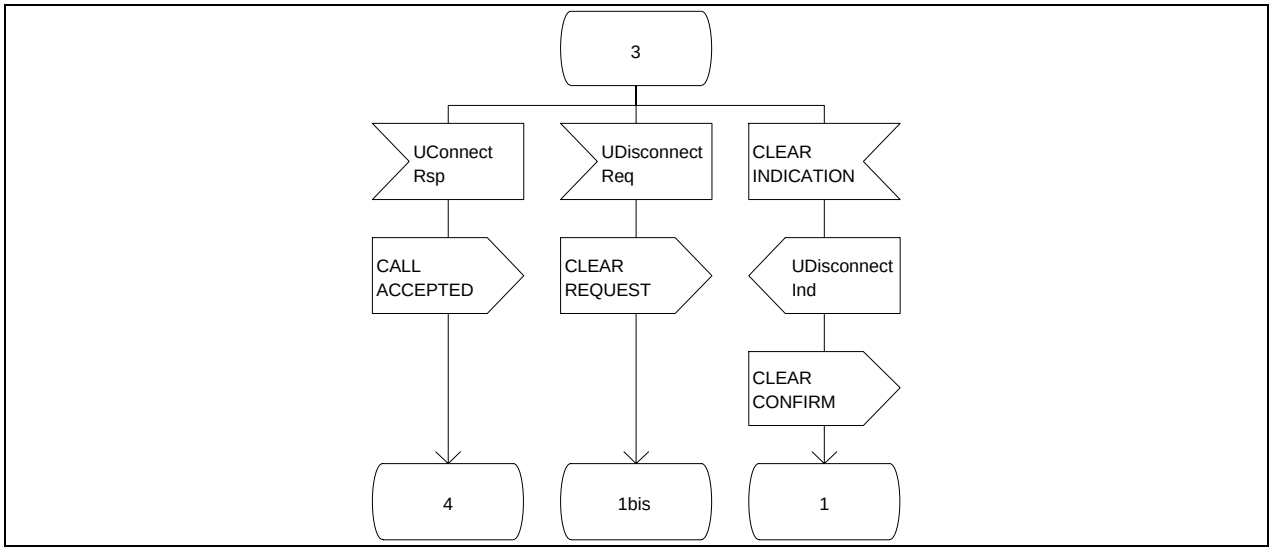


Figure E.40: INCOMING CONNECTION PENDING

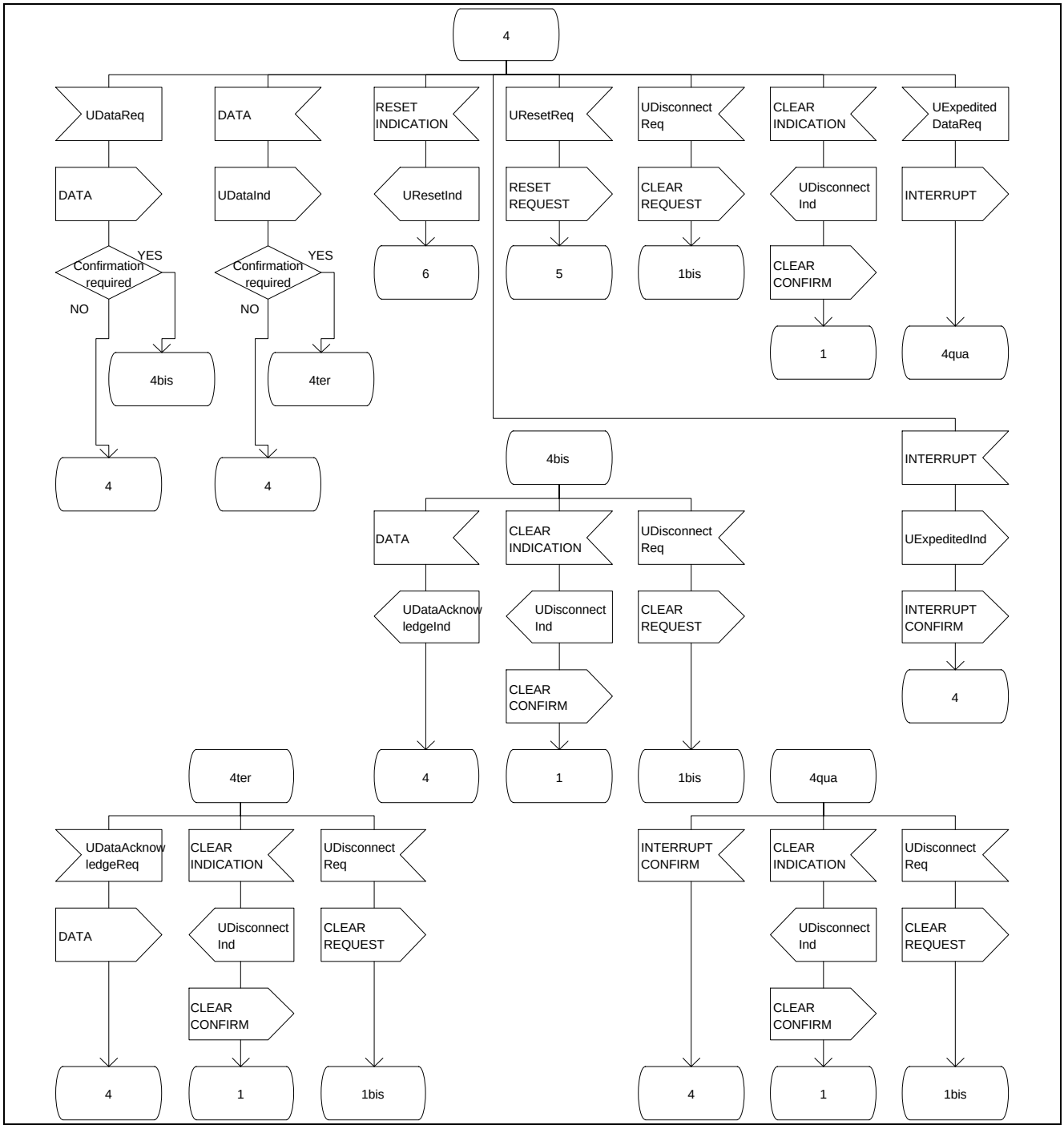


Figure E.41: DATA TRANSFER READY

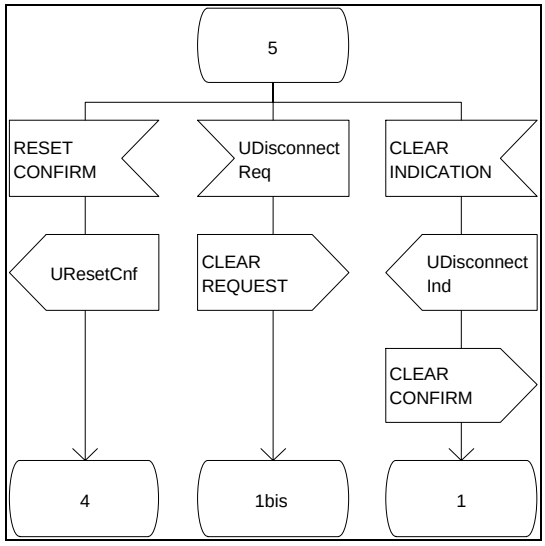


Figure E.42: PUF INVOKED RESET PENDING

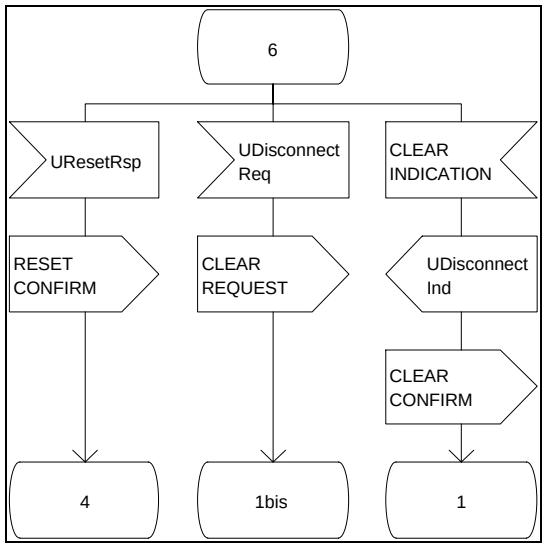


Figure E.43: NETWORK INVOKED RESET PENDING

E.2 Information provided by the NAF

The provision of items in messages can vary. The following conventions apply to the provision of elements by the NAF to the PUF:

- Mandatory parameters
  These items shall be provided.

- Conditional parameters
  The condition determines if they shall be provided.

- Optional parameters
  These items may or may not be provided depending on their availability to the NAF.

E.3 Suspending/resuming calls

The NAF is required to manage the mapping of NCOID to the call identity which is required by the network when resuming a call. Once the connection is resumed the NAF should ensure the mapping of the NCOID to the Call reference, which may have changed, on the network side.

E.4 Error management

The error indication provided to the PUF only contains sufficient information for the PUF to judge if it is worth continuing with a particular action or not. It is envisaged that more detailed information concerning a particular error will be reported by the NAF, in a NAF specific manner. For example, a NAF may choose to implement an error log in the form of a file. This file is where it records detailed information concerning a particular error. This provides the information required to debug a PUF which is under development.

The following subclauses provide guidance as to the conditions under which the NAF should return a particular error to the PUF.

For messages, in the case of parameters that are repeated where repetition is not allowed, only the first valid number of repetitions of the parameter are processed, further repetitions are ignored.

If an optional parameter is given by the network, the NAF is in charge of providing it to the PUF in the relevant message.

E.4.1 Function return codes

The description of the conditions under which these should be issued is described in subclause 5.9.5.

E.4.2 Administration Plane

The description of the conditions under which these should be issued is described in subclause 5.9.6. For the ACreateNCOReq the number of possible parameter combinations makes checking complex. It should be approached in the order shown in table E.17.

Table E.17: Checking of ACreateNCOReq message

Parameter	Test	Action
All parameters	Not allowed	InvalidParameter error
	All valid	Continue
NCOType	missing	MissingParameter error
	invalid length	InvalidParameterLength error
	Invalid value	InvalidNCOType error
	Valid value	Continue
Direction	missing	MissingParameter error
	invalid length	InvalidParameterLength error
	Invalid value	InvalidDirectionType error
	Valid value	Continue
AttributeName	missing	AttributeNameMissing error
	invalid length	InvalidParameterLength error
	invalid	AttributeNameError error
	correct	Continue
Attribute or address content	missing	MissingParameter error
	invalid length	InvalidParameterLength error
	invalid	InvalidContent error
	correct	Continue
GroupID	present but not required	GroupIDError error
	missing	GroupIDError error
	invalid length	InvalidParameterLength error
	invalid value	InvalidGroupID error
	correct	Continue
RequestID (if present)	invalid length	InvalidParameterLength error
	present	Process message

E.4.3 Control Plane

The errors returned in the Cause parameter match those in the ETS 300 102 [2] cause information element. This allows the NAF to pass information from the cause information element into the cause parameter. If this is done the NAF should map any information element values to parameter values as defined in annex B.

The following errors should be generated by the NAF as a result of checking parameters on messages passed from PUF to NAF.

Table E.18: Control Plane Cause parameter matching

Value	ETS 300 201 [1] Meaning	PCI Meaning	Generated by	When received from ISDN Processed by
1	Unallocated (unassigned) number		ISDN	PUF
2	No route to specified transit network		ISDN	NAF (note 1)
3	No route to destination		ISDN	PUF
6	Channel not acceptable		ISDN	NAF (note 1)
7	Call placed on an already establisheD-channel		ISDN	PUF
16	Normal call clearing		ISDN	PUF
17	User busy		ISDN	PUF
18	No user responding		ISDN	PUF
19	No answer from user (user alerted)		ISDN	PUF
21	Call Rejected		ISDN	PUF
22	Address changed		ISDN	PUF
26	Non selected user clearing		ISDN	PUF
27	Destination out of order		ISDN	PUF
28	Invalid address format	Parameter has invalid address format	NAF, ISDN	PUF
29	Facility rejected	Facility is not provided by this NAF	NAF, ISDN	PUF
30	Response to STATUS ENQUIRY		ISDN	NAF
31	Normal unspecified		ISDN	PUF
34	No circuit/channel available	Temporarily no channel of requested type is available from this NAF	NAF, ISDN	PUF
38	Network out of order		ISDN	NAF (note 1)
41	Temporary failure		ISDN	NAF (note 1)
42	Switching equipment congestion		ISDN	PUF
43	Access information discarded		NAF, ISDN	PUF (note 3)
44	RequesteD-channel/circuit not available	No channel of requested type is available from this NAF	NAF, ISDN	PUF
47	Resource unavailable, unspecified	Requested external equipment is not available	NAF, ISDN	PUF
49	Quality of service unavailable		ISDN	PUF
50	Facility requested on Facility parameter is not subscribed		ISDN	PUF
57	Bearer Capability not authorised		ISDN	PUF
58	Bearer Capability not presently available	Service requested by BearerCap is not available. In use by another PUF	NAF, ISDN	PUF
63	Service or option not available, unspecified		ISDN	PUF
65	Service requested by Bearer Capability is not implemented	Service requested by BearerCap Parameter is not provided by NAF	NAF, ISDN	PUF
66	Channel Type not implemented	NAF does not support this type of channel	NAF, ISDN	PUF
69	Facility requested is not implemented	NAF does not support this facility	NAF, ISDN	PUF
70	Only restricted digital information bearer capability is available		ISDN	NAF (note 1)
79	Service or option not implemented, unspecified		ISDN	PUF
81	Invalid call reference	Invalid NCOID	NAF, ISDN	NAF (note 1)
82	IdentifieD-channel does not exist	Identified permanent channel is not defined	NAF, ISDN	NAF (note 1)
83	A suspended call exists but this call identity does not		ISDN	NAF (note 1)
85	No call suspended	NCOID does not identify a suspended connection	NAF, ISDN	NAF (note 1)

(continued)

Table E.18 (concluded): Control Plane Cause parameter matching

Value	ETS 300 201 [1] Meaning	PCI Meaning	Generated by	When received from ISDN Processed by
86	Call having requested call identity has been cleared		ISDN	NAF (note 1)
88	Incompatible destination		ISDN	PUF
91	Invalid transit network selection		ISDN	NAF (note 1)
95	Invalid message, unspecified		ISDN	NAF (note 1)
96	Mandatory parameter is missing	Mandatory parameter is missing	NAF, ISDN	NAF (note 1)
97	Message Identifier non-existent or not implemented on this network	Message Identifier non-existent or not implemented on this NAF	NAF, ISDN	NAF (note 1)
98	Message not compatible with call state or message identifier non-existent or not implemented.	Message not compatible with NCO state or message identifier non-existent or not implemented.	NAF, ISDN	NAF (note 1)
99	Invalid parameter	Invalid parameter	NAF, ISDN	NAF (note 1)
100	Invalid parameter contents	Invalid parameter contents	NAF, ISDN	NAF (note 1)
101	Message not compatible with current state	Message not compatible with current state	NAF, ISDN	NAF (note 1)
102	Recovery on timer expiry		ISDN	NAF (note 2)
111	Protocol Error, unspecified		ISDN	NAF (note 1)
127	Interworking, unspecified		ISDN	PUF

NOTE 1: Where cause values are processed by the NAF. The NAF should attempt error recovery. If it fails to recover it should indicate to registered PUFs that it is no longer available by the use of the NAFNotAvailable error code.
NOTE 2: NAF should take appropriate action.
NOTE 3: In the case of ISDN generating this cause, it is the responsibility of the NAF to map information elements to parameter types in any diagnostic information supplied to PUF.

Table E.19 shows the order of checking for CConnectReq. Information is taken from the CConnectReq message plus the attribute and address sets associated with the mandatory network connection identifier. The table assumes that the initial checking of the message has taken place.

Table E.19: Checking of CConnectReq message

Parameter	Test	Action
Message state	invalid	CStatusInd Cause parameter value = 101 Diagnostics = MessageID
	valid	Combine parameters from attribute set, address set and CConnectReq message, continue
Mandatory Parameters	BearerCap missing	CDisconnectInd Cause parameter value = 96 Diagnostics = BearerCap
	BearerCap Service is not X.25 and CalledNumber missing	CDisconnectInd Cause parameter value = 96 Diagnostics = CalledNumber
	All present	Continue
BearerCap Parameter Content	Invalid contents	CDisconnectInd Cause parameter value = 100 Diagnostics = BearerCap
	Service not available from NAF	CDisconnectInd Cause parameter value = 65
	correct	Continue
CalledNumber Parameter Content (if present)	invalid contents	CDisconnectInd Cause parameter value = 100 Diagnostics = CalledNumber
	correct	Continue
Unrecognised Parameters	present	CDisconnectInd Cause parameter value = 99 Diagnostics = Parameter Type of unrecognised parameter
	not present	Continue
Optional Parameter Content Error	present	CStatusInd Cause parameter value = 100 Diagnostics = Parameter Type of parameter in error Continue (ignore parameter)
	not present	Process Message

E.5 NAF configuration

The following subclause contains information concerning NAF configuration. This subclause is provided to assist NAF developers and is not intended to be a comprehensive list of configurable items.

E.5.1 Global configuration

Table E.20: Global Configuration

Parameter	Suggested Default	Comment
Number of PUFs supported	8	

E.5.2 System configuration parameters

Table E.21: System configuration parameters

Parameter	Suggested default	Comment
DMA		DMA number used by the adapter
I/O address		I/O address used by the adapter
IRQ		IRQ used by the adapter
DRAM		Double RAM access address shared between the adapter and the host environment. This parameter may also contain the size of the frame to be used by the adapter

E.5.3 Control Plane configuration

Table E.22: Control Plane configuration

Parameter	Suggested Default	Comment
Number of D-channels	1	
D-channel definitions - type of network - automatic - fixed + number - frame window size (K) - N200 - N201 - N 202 Timers, - T200 - T201 - T202 - T203	1	
Number of B-channels	2	
Number of Permanent B-channels	0	
List of permanent B-channel identifiers	1..256	
Number of permanent (SAPI 16) D-channels For each D-channel - automatic - fixed + number - same as signalling	0	

E.6 Buffer management

Buffers passed by the PUF to the NAF are copied by the NAF into internal space as soon as provided. So, the buffers may be reused by the PUF immediately after the function returns.

The exact instant when the message is processed is dependent on the NAF and is outside the scope of this ETS.

In the PUF to NAF direction, the message and the associated data, if any, are provided together, in one step. If one of the messages or the data buffers is too small to contain, respectively, a complete message or the data information, the NAF shall return an error and the message shall not be provided to the PUF. To help the PUF, the size of the biggest message is established during the registration phase. The size of a data buffer is closely dependent of the type of connection and its protocol. The PUF has to refer to the User Plane protocol initialisation to get the correct value of the longest data packet.

If a NAF needs new internal buffer is, it is in charge of this action. This can be achieved via a configuration operation, provided by the NAF manufacturer, which is outside the scope of this ETS. The NAF manufacturer may describe how the operation can be realised and which consequences are expected.

E.7 NAF development user consideration

The main body of this ETS contains the description of Profile A from the PUF point of view. Following this approach, some points, not directly related to the PUF, which have an impact on the development of the NAF are not described. These points may be of interest for the NAF development and are, therefore, described in this annex. It gives guidelines for the development of the NAF in accordance with the main body of this ETS.

Consider this subclause from the following viewpoints:

- this annex gives additional points. The NAF has to be implemented using this ETS. It should implement Profile A in such a way that the functionality described is provided;

- the main body of this ETS should be given priority if there is anything not clear in this subclause or the interpretation between the main body of this ETS and this subclause is different;

- this subclause does not try to impose any constraints on the implementation of the NAF. The objective is to give guidelines as to how the NAF can be developed to be in line with this ETS.

E.7.1 User Plane error management

The error processing for this plane is defined for each protocol in the relevant subclause.

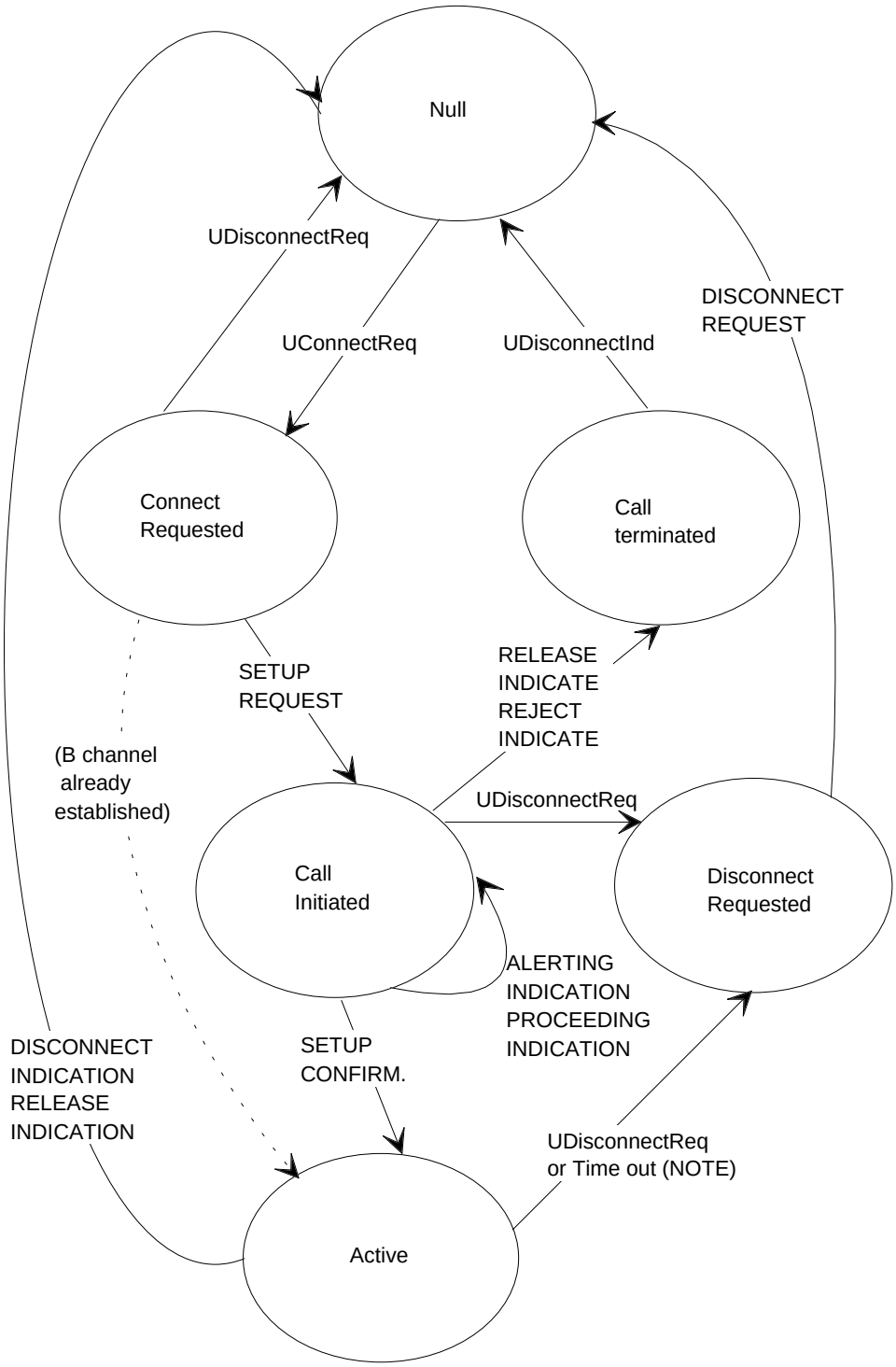
E.7.2 NAF configuration

Global configuration, system configuration and Control Plane configuration are provided in clause E.5 of this annex.

User Plane configuration can be found in subclauses of this annex, depending on the protocol.

E.7.3 Co-ordination function - outgoing User Plane call

The following state diagram shows the establishment of the Control Plane connection. The states indicated are internal to the NAF.



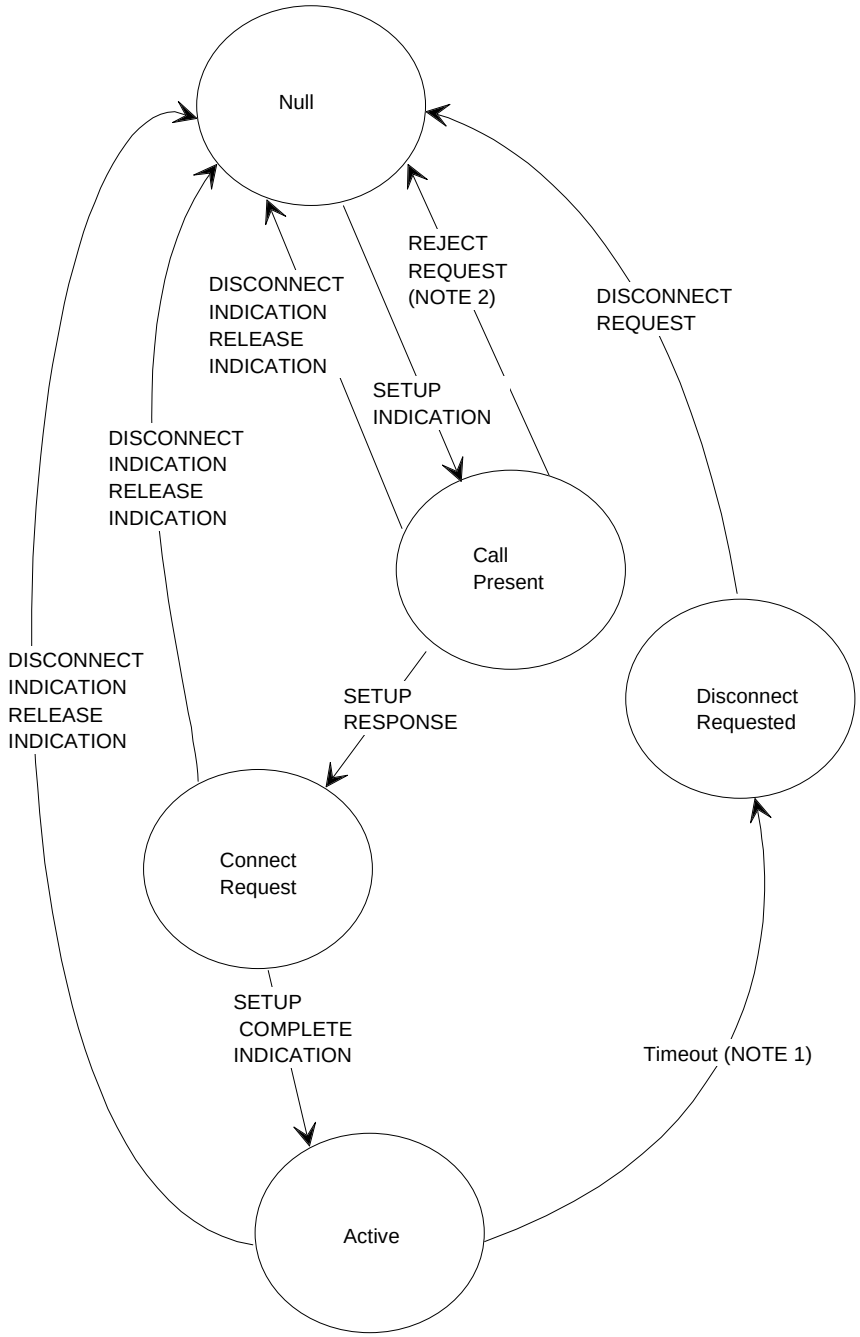
Remarks:
- events in upper case are primitives described in ETS 300 102 [2];
- events in mixed case are PCI User Plane messages;
- the states shown are internal to the NAF.

NOTE: Whether the NAF disconnects the ISDN connection following the disconnection of the last User Plane connection on the ISDN connection or sets a time-out is a NAF design consideration.

Figure E.44: Co-ordination function - outgoing call and channel establishment

E.7.4 Co-ordination function - incoming ISDN call

The following state diagram (figure E.45) shows the establishment of the Control Plane connection. The states indicated are internal to the NAF.



Remarks:
- events in upper case are primitives described in ETS 300 102 [2];
- the states shown are internal to the NAF.

NOTE 1: Whether the NAF disconnects the ISDN connection following the disconnection of the last User Plane connection on the ISDN connection or sets a time-out is a NAF design consideration.
NOTE 2: The NAF may reject the ISDN connection request.

Figure E.45: ISDN incoming call and co-ordination function

E.8 User protocols key information

The following table summarises key information to make use of User Plane protocols. Values in parenthesis are decimal coded.

PROTOCOL	NCOType	UProtocol	UDirection	Co-ordination Function
Transparent access	. C/U	. NULL (4) . NULL (8) . Transparent access (1)	"both" or absent	NO
V.110	. C/U	. NULL (4) . V.110 (6 or 7) . (...)	"both" or absent	NO
PPP	. C/U	. NULL (4) . PPP (4) . (...)	"both" or absent	NO
SDLC	. C/U	. NULL (4) . SDLC (5) . (...)	"both" or absent	NO
HDLC	. C/U	. NULL (4) . HDLC (2 or 3) . (...)	"both" or absent	NO
ISO 7776	. C/U	. NULL (4) . ISO 7776 (1) . (...)	"both" or absent	NO
T.70	. C/U	. T.70 (3) . (...) . (...)	"both" or absent	NO
ISO 8208	. C/U . U3 . U3G	. ISO 8208 (2) . (...) . (...)	used	YES
ETS 300 080	. C/U . U3 . U3G	. T.90 (1) . ISO 7776 (1) . (...)	used	YES
NOTE: When (...) is used, it means that there is no fixed value.				

Annex F (normative): Profile A implementation description

F.1 Copyright release for Profile A implementation description

Notwithstanding the provisions of the copyright clause related to the text of the present ETS, ETSI grants users of this ETS to freely reproduce the Profile A implementation description in this annex so that it can be used for its intended purposes and may further publish a completed ICS.

F.2 Introduction

This annex contains the Profile A implementation description. The Profile A implementation description lists all mandatory, conditional and optional items of Profile A of this specification relating to the exchange mechanism and the supported messages. It shall be used in the process of evaluating a particular implementation when claiming conformance to or support of Profile A of this specification. The implementation which claims conformance can either be a PUF or a NAF. For the PUF, the Profile A implementation description indicates if it uses the item. For the NAF this indicates if the item is supported.

To evaluate conformance of a particular implementation, a statement of which capabilities and options have been implemented should be given. This annex contains such a statement.

The supplier of an implementation which is conforming to Profile A of this ETS shall complete a copy of the Profile A implementation description and shall provide information necessary to identify both the supplier and the implementation.

F.3 Profile A implementation description cover page

F.3.1 Identification of the Profile A implementation description

Profile A implementation description serial no:
Date:

F.3.2 Identification of implementation

Name :
Version :
Special configuration :
Other information :

F.3.3 Identification of the system supplier

Name	:	Contact	:
Street	:	Phone no	:
City	:	Telex no	:
Country	:	Fax no	:

F.3.4 Global statement of conformance

Are all mandatory features implemented? (Yes or No) :

Answering "No" to this question indicates non-conformance to the Profile A interface specification. Non-supported mandatory capabilities shall be indicated in the Profile A implementation description, with an explanation of why the implementation is non-conformant.

F.4 Instructions for completing the Profile A implementation description

Each line within the Profile A implementation description which requires implementation details to be entered is numbered at the left hand edge of the line. This numbering is included as a means of uniquely identifying all possible implementation details within the Profile A implementation description.

The N/P column in this annex separates the capabilities for the Network Access Facility (NAF) and the PCI User Facility (PUF).

N Network Access Facility (NAF)

P PCI User Facility (PUF)

The D column in this annex reflects the Definition of the items in this ETS. Each entry in this column is chosen from the following list:

M Mandatory support is required;

O Optional support is permitted. If implemented, it shall conform to this ETS;

C Conditional support. The support of this element is subject to a condition which is described in the note column.

The I column in this annex describes the actual capabilities of the Implementation and shall be completed by the supplier using a symbol chosen from the following list:

Y item is implemented (subject to stated constraints);

N item is not implemented;

- item is not applicable.

The ref./note column in this annex contains the references to the location in the main body of this ETS where the items are described or a note explaining why the item is conditional.

The following abbreviations are used in the headings of this Profile A implementation description:

P/N PCI User Facility (PUF)/Network Access Facility (NAF).

D Defined.

I Implemented.

ref. reference.

F.5 Exchange mechanism

	Item of Profile A	N/P	D	I	ref./note
1	PciGetHandles	N	O	[ ]	5.3.1.2
		P	O	[ ]	5.3.1.2
2	PciGetProperty	N	M	[ ]	5.3.1.3
		P	O	[ ]	5.3.1.3
3	PciRegister	N	M	[ ]	5.3.1.4
		P	M	[ ]	5.3.1.4
4	PciPutMessage	N	M	[ ]	5.3.3.6
		P	M	[ ]	5.3.3.6
5	PciGetMessage	N	M	[ ]	5.3.3.7
		P	M	[ ]	5.3.3.7
6	PciSetSignal	N	M	[ ]	5.3.3.8
		P	O	[ ]	5.3.3.8
7	PciDeRegister	N	M	[ ]	5.3.2.1
		P	M	[ ]	5.3.2.1

F.6 Administration Plane

	Item of Profile A	N/P	D	I	ref./note
8	Administration Plane message class 1	N	M	[ ]	5.4
	Basic class	P	M	[ ]	5.4
9	Administration Plane message class 2	N	O	[ ]	5.4
	Security features	P	O	[ ]	5.4
10	Administration Plane message class 3	N	O	[ ]	5.4
	Manufacturer specific features	P	O	[ ]	5.4
11	Administration Plane message class 4	N	O	[ ]	5.4
	Change NCO parameter	P	O	[ ]	5.4

F.7 Control Plane

	Item of Profile A	N/P	D	I	ref./note
12	Control Plane message class 1	N	M	[ ]	5.5
	Basic class	P	C	[ ]	5.5 Use of co-ordination function
13	Control Plane message class 2	N	O	[ ]	5.5
	Overlap sending	P	O	[ ]	5.5
14	Control Plane message class 3	N	O	[ ]	5.5
	User-to-user information transfer	P	O	[ ]	5.5
15	Control Plane message class 4	N	O	[ ]	5.5
	Call adjournment for telephony	P	O	[ ]	5.5
16	Control Plane message class 5	N	O	[ ]	5.5
	Facility invocation	P	O	[ ]	5.5
17	Control Plane message class 6	N	O	[ ]	5.5
	External Equipment handling	P	O	[ ]	5.5
18	Control Plane message class 7	N	O	[ ]	5.5
	Additional information	P	O	[ ]	5.5
19	Control Plane message class 8	N	O	[ ]	5.5
	DTMF	P	O	[ ]	5.5

F.8 User Plane

	Item of Profile A	N/P	D	I	ref./note
20	User Plane message class 1	N	M	[ ]	5.6
	Basic class	P	M	[ ]	5.6

F.9 User Plane protocols

	Item of Profile A	N/P	D	I	ref./note
21	Network layer protocol according to ETS 300 080 [2]	N	M	[ ]	5.6.4.1
		P	O	[ ]	5.6.4.1
22	Network layer protocol according to ISO/IEC 8208	N	M	[ ]	5.6.4.1
		P	O	[ ]	5.6.4.1
23	Transparent User Plane protocol	N	M	[ ]	5.6.2.1
		P	O	[ ]	5.6.2.1
24	Network layer protocol according to network layer of T.70	N	O	[ ]	5.6.4.2
		P	O	[ ]	5.6.4.2
25	Network layer protocol using Null layer 3 with access to ISO 7776 on layer 2	N	O	[ ]	5.6.3.1
		P	O	[ ]	5.6.3.1

(continued)

F.9 User Plane protocols

	Item of Profile A	N/P	D	I	ref./note
26	Network layer protocol using Null layer 3 with transparent access to HDLC framing	N	O	[ ]	
		P	O	[ ]	
27	Network layer protocol using Null layer 3 with transparent access to HDLC framing reporting error	N	O	[ ]	5.6.3.3
		P	O	[ ]	5.6.3.3
28	Network layer protocol using Null layer 3 with transparent access to PPP	N	O	[ ]	5.6.3.4
		P	O	[ ]	5.6.3.4
29	Network layer protocol using Null layer 3 with transparent access to SDLC	N	O	[ ]	5.6.3.5
		P	O	[ ]	5.6.3.5
30	Network layer protocol using Null layer 3 with transparent access to V.110	N	O	[ ]	5.6.3.6
		P	O	[ ]	5.6.3.6
31	Network layer protocol using Null layer 3 with transparent access to V.120	N	O	[ ]	5.6.5
		P	O	[ ]	5.6.5
32	Network layer protocol using Null layer 3 with transparent access to T.30	N	O	[ ]	5.6.6
		P	O	[ ]	5.6.6

F.10 Miscellaneous features

	Item of Profile A	N/P	D	I	ref./note
33	Transparent coding of facility information element.	N	O	[ ]	5.2.1.3.3.2, 5.7.32
		P	O	[ ]	5.2.1.3.3.2, 5.7.32

Annex G (normative): Static attribute content for the Control Plane

This annex contains a complete description of the static attributes that a NAF shall provide for the Control Plane. Rules to establish these attributes are related to the protocol requirement. User Plane related static attributes should be found in the subclause 5.6 of this ETS.

G.1 Generic circuit bearer service

G.1.1 Speech

Name : "SPEECH_A-LAW"
BearerCap : 80 90 A3
LLC : Not used
HLC : Not used

Name : 'SPEECH_μ-LAW'
BearerCap : 80 90 A2
LLC : Not used
HLC : Not used

G.1.2 Unrestricted digital information

Name : 'UNRESTRICTED'
BearerCap : 88 90
LLC : Not used
HLC : Not used

G.1.3 Restricted digital information

Name : 'UNRESTRICTED/56'
BearerCap : 88 90 01 8F
LLC : Not used
HLC : Not used

G.1.4 3,1 Khz audio information transfer

Name : 'AUDIO_A-LAW'
BearerCap : 90 90 A3
LLC : Not used
HLC : Not used

Name : 'AUDIO_μ-LAW'
BearerCap : 90 90 A2
LLC : Not used
HLC : Not used

G.1.5 Packet mode bearer service

Name : 'D_CHANNEL_HDL'
BearerCap : 88 C0 C6 E6
LLC : Not used
HLC : Not used

G.1.6 Teleservices

Name : 'TELEPHONYA_LAW'
BearerCap : 80 90 A3
LLC : Not used
HLC : Standard = 0
Identification = 1

Name : 'TELEPHONYμ_LAW'

BearerCap : 80 90 A2
LLC : Not used
HLC : Standard = 0
Identification = 1

Name : 'TELEFAX_G4'
BearerCap : 88 90
LLC : Depending on Terminal Equipment: octet 3a. Not used: octet 4 and 5.
Octet 6 (layer 2) = 0D (13)
Octet 7 (layer 3) = 07
HLC : Standard = 0
Identification = 21 (33)

Annex H (informative): Operating System implementation coding samples for Profile A

H.1 DOS Operating System implementation coding samples

These samples present a way to implement the exchange mechanism function call from the PUF point of view.

```
/*****************************************************************************************
This library code may be linked to a PUF to allow uniform access to multiple NAFs. The access to
the different NAFs by use of an unique ExID is achieved by the use of a local table, which allows
MAX_EXID entries.
*****************************************************************************************/
/*
 * Include files
 */
#include <dos.h>
#include <fcntl.h>
#include <memory.h>
#include <malloc.h>
#include <stdio.h>

/*
 * General typedefs
 */
typedef void            ( *   PFRV) ();         /* Pointer to Function Returning Void          */
typedef short int       ( far *  FPFRS) ();          /* Far Pointer to Function Returning Short
      */
typedef void            ( far *  FPFRV) ();          /* Far Pointer to Function Returning Void */
typedef int       ( far *  FPFRI) ();          /* Far Pointer to Function Returning Int */

/*
 * Mapping of generic type definitions
 */

typedef short int            PCI_INTEGER;
typedef char far  *          PCI_BYTEARRAY;
typedef short int            PCI_EXID;
typedef FPFRI                PCI_HANDLE;
typedef FPFRV                PCI_PROCEDURE;

/*
* Definition of function codes
*/

#define PCIGETPROPERTY       (short) (1)
#define PCIREGISTER          (short) (2)
#define PCIDEREGISTER        (short) (3)
#define PCIPUTMESSAGE        (short) (4)
#define PCIGETMESSAGE        (short) (5)
#define PCISETSIGNAL         (short) (6)

/*
 * Error definitions
 */
#define E_DEVICE_DRIVER_NOT_FOUND        128
#define E_DEVICE_DRIVER_CONTROL          128
#define E_NAF_NOT_FOUND            130
#define E_NAF_INVALID_ADDRESS      130
#define E_TOO_MANY_ASSOCIATIONS          133
#define E_INVALID_EXCHANGE_ID            136

/*
 * Other definitions
 */

#define SUCCESS                  0
#define MAX_EXID             32 /* allow 32 PUF_NAF associations */

/*
 * Structures
 */

struct pci_mpb {
      PCI_INTEGER MessageID;
      PCI_INTEGER MessageMaximumSize;
      PCI_INTEGER MessageActualUsedSize;
      PCI_INTEGER DataMaximumSize;
      PCI_INTEGER DataActualUsedSize;
};
```

```
struct pci_register {                    /* structure containing registering info */
      PCI_INTEGER PUFVersion;            /* optional: give PUF version */
      PCI_INTEGER PUFType;               /* optional: give PUF type */
      PCI_INTEGER MaxMsgSize;      /* return: max size of a message */
};

struct pci_opsys {                              /* structure containing registering info */
      short int         MaxNCOCount;            /* optional: give max count of NCOs */
      short int         MaxPacketSize;          /* optional: give expected max size and */
      short int         MaxPacketCount;         /* max count of packets to buffer */
      long int          AddBufferSize;          /* optional: give to NAF size and */
      void far *        AddBufferSpace;         /* pointer to additional buffer */
      long int          BufferNeeded;           /* return: amount of add buffer needed */
};

struct loc_exid_map {                    /* locally used structure for ExIDs */
      PCI_HANDLE  pci_handle;
      PCI_EXID    exchange_id;
};

/*
* Functional constants
*/
const char PCIsign[8]="ISDN PCI";

/*
 * Local variables
 */
static struct loc_exid_map _exid_map[MAX_EXID];      /* table holding MAX_EXID ExID entries */
static short int _exid_cnt = MAX_EXID;               /* count of free places inside ExID table */
```

```
/*************************************************************************
PciGetHandles:    Asks the "PCIDD$" device driver for a list of available
      PCI-Handles (NAF entry points).
      Returns available PCI-Handles into the given PCIHandles buffer.
      The maximum amount of PCI-Handles requested is given in MaxHandles.
      Function will fail, if MaxHandles is less than the Handles available in the driver.
***/
short int PciGetHandles (     short int MaxHandles,
                FPFRI * PCIHandles,
                short int * ActualHandles )
{
short int fildes;       /* file descripror */
union _REGS regs;
struct _SREGS segregs;

      /* open the driver */
if (_dos_open ("PCIDD$", _O_RDWR, &fildes) != SUCCESS)
      return E_DEVICE_DRIVER_NOT_FOUND;        /* device driver not accessible; return error */

      /* prepare IOCTL read from device driver */
_segread (&segregs);
segregs.ds = FP_SEG (PCIHandles);                    /* set-up segment address */
regs.x.dx = FP_OFF (PCIHandles);                     /* and offset */
regs.x.cx = MaxHandles * sizeof(PCI_HANDLE);
regs.x.bx = fildes;                                  /* set dos file handle */
regs.x.ax = 0x4402;                             /* IOCTL read from character device */

      /* issue IOCTL read from device driver */
_intdosx (&regs, &regs, &segregs);

      /* close the driver */
_dos_close (fildes);

      /* check for error */
if (regs.x.cflag & 1)                           /* check processors carry flag */
      return E_DEVICE_DRIVER_CONTROL;           /* error has occured; return error */

      /* Successful operation. Compute count of PCI-Handles received */
*ActualHandles = regs.x.ax / sizeof(PCI_HANDLE);

return SUCCESS;
}     /* End of PciGetHandles() */
```

```
/*************************************************************************
PciGetProperty:   Asks the NAF for it's properties, which is a list of TLV coded topics.
      Returns the properties into the given Property buffer.
      The maximum size of the Property buffer is given in MaximumSize.
      Function will fail, if MaximumSize is less than the size of the Property the
      NAF can deliver.
***/

short int PciGetProperty ( FPFRI PCIHandle,
      short int MaximumSize,
      char * Property,
      short int * ActualSize )
{
register short int error;
unsigned char far * signature;

/* Check if NAF is available */
if ( PCIHandle == NULL)
      return E_NAF_INVALID_ADDRESS; /* NAF inaccessible, invalid address */

/* compute address of signature and check it */
signature = (unsigned char far *) PCIHandle - sizeof(PCIsign);
if ( _fmemcmp (PCIsign, signature, sizeof(PCIsign) != SUCCESS))
      return E_NAF_NOT_FOUND; /* NAF inaccessible invalid signature */

/* Call the NAF to obtain the property information */
error = (*PCIHandle) (  PCIGETPROPERTY,
      MaximumSize,
      (char far *) Property,
      (short int far *) ActualSize );
return error;
}     /* End of PciGetProperty() */

/*************************************************************************
PciRegister:      Tries to associate calling PUF with selected NAF.
      Delivers the ExID, which has to be used in subsequent calls.
      Two structures have to be provided by the calling PUF:
      -     The PCIRegisterInfo and
      -     the PCIOpSysInfo structure.
***/

short int PciRegister ( FPFRI PCIHandle,
      struct pci_register * PCIRegisterInfo,
      struct pci_opsys * PCIOpSysInfo,
      short int * ExID )
{
register short int error;
register short int exchange_id;
unsigned char far * signature;
struct loc_exid_map *exid_map;      /* dynamic pointer to local _exid_map tab */

/* Check if NAF is available */
if ( PCIHandle == NULL)
      return E_NAF_INVALID_ADDRESS; /* NAF inaccessible, invalid address */

/* compute address of signature and check it */
signature = (unsigned char far *) PCIHandle - sizeof(PCIsign);
if ( _fmemcmp (PCIsign, signature, sizeof(PCIsign) != SUCCESS))
      return E_NAF_NOT_FOUND;       /* NAF inaccessible invalid signature */

/* check if there is still room in our local _exid_map table */
if ( ! _exid_cnt)
      return E_TOO_MANY_ASSOCIATIONS;     /* Indicate table exhausted */

/* Call the NAF to inform it of a new association PUF */
error = (*PCIHandle) (  PCIREGISTER,
      (struct pci_register far *) PCIRegisterInfo,
      (struct pci_opsys far *) PCIOpSysInfo,
      (short int far * ) ExID );
if ( ! error)
      {
      /* Association was successful; record it in local table */
      exchange_id = 0;
      exid_map = &_exid_map[0];                /* setup pointer into local _exid_map table */
      while (exid_map->pci_handle)
            {
            exid_map++;
            exchange_id += 1;
            }
      exid_map->exchange_id = *ExID;
      exid_map->pci_handle = PCIHandle;
      *ExID = exchange_id;                /* compute and set Exchange-ID */
      _exid_cnt -= 1;
      }
```

```
return error;
}     /* End of PciRegister() */

/*************************************************************************
PciDeregister:    Terminates an existing association wit a NAF.
      The ExID of an existing association has to be provided.
***/

short int PciDeregister (  PCI_EXID ExID )
{
register short int error;
struct loc_exid_map *exid_map;                /* dynamic pointer to local _exid_map tab */

/* Check if ExID is valid and setup pointer into local _exid_map table */
exid_map = &_exid_map[ExID];
if (ExID < 0 || ExID >= MAX_EXID || ! exid_map->pci_handle)
      return E_INVALID_EXCHANGE_ID;

/* Call the NAF to inform it of the end of the association */
error = (*exid_map->pci_handle) (   PCIDEREGISTER,
      exid_map->exchange_id );

/* delete association from local table */
exid_map->pci_handle = NULL;
_exid_cnt += 1;

return error;
}     /* End of PciDeregister() */
```

```
/*****************************************************************************
PciPutMessage:    Transfers a Message and associated Data to the NAF.
***/

short int PciPutMessage (  short int ExID,
      struct pci_mpb * PCIMPB,
      char * Message,
      char * Data )
{
register short int error;
struct loc_exid_map *exid_map;      /* dynamic pointer to local _exid_map tab */

/* Check if ExID is valid and setup pointer into local _exid_map table */
exid_map = &_exid_map[ExID];
if (ExID < 0 || ExID >= MAX_EXID || ! exid_map->pci_handle)
      return E_INVALID_EXCHANGE_ID;
/* Call the NAF and provide the message */
error = (*exid_map->pci_handle) (   PCIPUTMESSAGE,
      exid_map->exchange_id,
      (struct pci_mpb far *) PCIMPB ,
      (char far *) Message,
      (char far *) Data );
return error;
}     /* End of PciPutMessage() */


/*****************************************************************************
PciGetMessage:    Receives a Message and associated Data from the NAF.
***/
short int PciGetMessage (  short int ExID,
      struct pci_mpb * PCIMPB ,
      char * Message,
      char * Data )
{
register short int error;
struct loc_exid_map *exid_map;      /* dynamic pointer to local _exid_map tab */

/* Check if ExID is valid and setup pointer into local _exid_map table */
exid_map = &_exid_map[ExID];
if (ExID < 0 || ExID >= MAX_EXID || ! exid_map->pci_handle)
      return E_INVALID_EXCHANGE_ID;

/* Call the NAF and receive the message */
error = (*exid_map->pci_handle) (   PCIGETMESSAGE,
      exid_map->exchange_id,
      (struct pci_mpb far *) PCIMPB ,
      (char far *) Message,
      (char far *) Data );

return error;
}     /* End of PciGetMessage() */
```

```
/*************************************************************************
PciSetSignal: Hands the address of a SignalProcedure to the NAF.
      The SignalProcedure then will receive notification on communication
      events (i.e. Message available for retrieval)
***/

short int PciSetSignal (   short int ExID,
      short int Signal,
      PFRV SignalProcedure )
{
register short int error;
struct loc_exid_map *exid_map;     /* dynamic pointer to local _exid_map tab */

/* Check if ExID is valid and setup pointer into local _exid_map table */
exid_map = &_exid_map[ExID];
if (ExID < 0 || ExID >= MAX_EXID || ! exid_map->pci_handle)
      return E_INVALID_EXCHANGE_ID;

/* Call the NAF to set the signal function */
error = (*exid_map->pci_handle) (   PCISETSIGNAL,
      exid_map->exchange_id,
      (FPFRV) SignalProcedure );

return error;
}     /* End of PciSetSignal() */
```

H.2 WINDOWS Operating System implementation coding samples

These samples present a way to implement the exchange mechanism function call from the PUF point of view.

The following code shows a sample implementation of PUF exchange functions for the Windows environment. The sample is illustrated using "C" language:

```
/* standard includes */
#include <windows.h>

/* Basic types */
typedef short           PCI_INTEGER;
typedef LPSTR           PCI_BYTEARRAY;
typedef LPSTR           PCI_HANDLE;
typedef struct {
      HINSTANCE         hDLLInstance;
      PCI_INTEGER       Exchange_Id;
      } PCI_EXID;
typedef void (far pascal *PCI_PROCEDURE)();


/* PCI Structures */
struct pci_mpb {
      PCI_INTEGER   MessageID;
      PCI_INTEGER   MessageMaximumSize;
      PCI_INTEGER   MessageActualUsedSize;
      PCI_INTEGER   DataMaximumSize;
      PCI_INTEGER   DataActualUsedSize;
      };
typedef struct pci_mpb PCI_MPB;

struct pci_register {                     /* structure containing registering info */
      PCI_INTEGER PUFVersion;       /* optional: give PUF version */
      PCI_INTEGER PUFType;                /* optional: give PUF type */
      PCI_INTEGER MaxMsgSize;       /* return: max size of a message */
};

struct pci_opsys {                        /* structure containing registering info */
      int   DummyParameter;         /* No specific requirement for WINDOWS */
};

/*
 * PCI defines
 */
#define PCI_HANDLE_LENGTH               128   /*  size  of  each  handle  in  the  buffer  from
PciGetHandles */
#define PCI_E_SUCCESS                   0
#define PCI_E_QUERY_ENTITY_NOT_AVAILABLE 128
#define PCI_E_INVALID_PCI_HANDLE        130
#define PCI_E_NAF_NOT_AVAILABLE         255
```

```
/*
////////////////////////////////////////////////////////////////////////////////
/// PciGetHandles()
*/
PCI_INTEGER far PASCAL PciGetHandles (   PCI_INTEGER MaxHandles,
                              PCI_HANDLE PCIHandles,
                              PCI_INTEGER far * ActualHandles)
      {
      int               nafNumber;
      int               nafFound;
      int               size;
      char              keyName[20];
      PCI_BYTEARRAY     buffer;

      buffer = PCIHandles;
      for (nafNumber = 1, nafFound = 0; nafNumber <= MaxHandles; nafNumber++)
            {
            wsprintf( keyName, "pciDriver%d", nafNumber);
            size = GetPrivateProfileString(
                        "DRIVERS",  /* Section name*/
                        keyName,    /* 'pciDriver'+1..n */
                        NULL,       /* No default string needed */
                        buffer,           /* Address where to put the result */
                        128,        /* Maxi. size for the result */
                        "PCI.INI"); /* INI FileName */
            if (size > 0)
                  {
                  nafFound++; /* One more NAF found */
                  buffer += 128;/* Next location for a PCIHandle (128 octets fixed size)*/
                  }
            }
      *ActualHandles = nafFound;
      }


/*
////////////////////////////////////////////////////////////////////////////////
/// PciGetProperty()
*/
PCI_INTEGER far PASCAL PciGetProperty (   PCI_HANDLE PCIHandle,
                              PCI_INTEGER MaximumSize,
                              PCI_BYTEARRAY Property,
                              PCI_INTEGER far * ActualSize)
      {
      PCI_INTEGER iReturnCode;
      HINSTANCE hDLLInstance;
      FARPROC lpfnGetProperty;

      /* load the NAF's DLL */
      hDLLInstance = LoadLibrary(PCIHandle);
      if (hDLLInstance < HINSTANCE_ERROR)
            return PCI_E_INVALID_PCI_HANDLE;    /* error in LoadLibrary */

      /* get the "PciGetProperty" entry point of the dll */
      lpfnGetProperty = GetProcAddress(hDLLInstance, "PciGetProperty");
      if (lpfnGetProperty == NULL)
            {
            FreeLibrary(hDLLInstance);
            return PCI_E_NAF_NOT_AVAILABLE;     /* error in GetProcAddress */
            }

      /* call the "PciGetProperty" entry point of the dll */
      iReturnCode = lpfnGetProperty(PCIHandle, MaximumSize, Property, ActualSize);

      /* free the DLL in any case */
      FreeLibrary(hDLLInstance);

      /* return with the DLL's return code */
      return iReturnCode;
      }


/*
////////////////////////////////////////////////////////////////////////////////
/// PciRegister()
///   The PCIOpSysInfo is kept for compatibility only
*/
PCI_INTEGER far PASCAL PciRegister (      PCI_HANDLE PCIHandle,
                              struct pci_register * PCIRegisterInfo,
                              struct pci_opsys * PCIOpSysInfo,
                              PCI_EXID far *ExID)
```

```
      {
      PCI_INTEGER iReturnCode;
      FARPROC lpfnRegister;
      HINSTANCE hDLLInstance;

      /* load the NAF's DLL */
      hDLLInstance = LoadLibrary(PCIHandle);
      if (hDLLInstance < HINSTANCE_ERROR)
            return PCI_E_INVALID_PCI_HANDLE;    /* error in LoadLibrary */

      /* put the DLL instance in ExID */
      ExID->hDLLInstance = hDLLInstance;

      /* get the "PciRegister" entry point of the dll */
      lpfnRegister = GetProcAddress(hDLLInstance, "PciRegister");
      if (lpfnRegister == NULL)
            {   /* error in GetProcAddress */
            FreeLibrary(hDLLInstance);
            return PCI_E_NAF_NOT_AVAILABLE;
            }

      /* call the "PciRegister" entry point of the dll */
      iReturnCode = lpfnRegister(PCIRegisterInfo, ExID);

      if (iReturnCode != 0)
            {   /* error in PciRegister: free the DLL */
            FreeLibrary(hDLLInstance);
            }

      /* return with the DLL's return code */
      return iReturnCode;
      }


/*
////////////////////////////////////////////////////////////////////////////////////////
/// PciDeRegister()
*/
PCI_INTEGER far PASCAL PciDeregister(PCI_EXID far *ExID)
      {
      PCI_INTEGER iReturnCode;
      FARPROC lpfnDeregister;

      /* get the "PciDeregister" entry point of the dll */
      lpfnDeregister = GetProcAddress(ExID->hDLLInstance, "PciDeregister");
      if (lpfnDeregister == NULL)                     /* error in GetProcAddress */
            return PCI_E_NAF_NOT_AVAILABLE;

      /* call the "PciDeRegister" entry point of the dll */

      iReturnCode = lpfnDeregister(ExID);

      /* free the DLL in any case */
      FreeLibrary(ExID->hDLLInstance);

      /* return with the DLL's return code */
      return iReturnCode;
      }


/*
////////////////////////////////////////////////////////////////////////////////////////
/// PciPutMessage()
*/
PCI_INTEGER far PASCAL PciPutMessage(     PCI_EXID far *ExID,
                              PCI_MPB far *PCIMPB ,
                              PCI_BYTEARRAY Message,
                              PCI_BYTEARRAY Data)
      {
      FARPROC lpfnPutMessage;

      /* get the "PciPutMessage" entry point of the dll */
      lpfnPutMessage = GetProcAddress(ExID->hDLLInstance, "PciPutMessage");
      if (lpfnPutMessage == NULL)   /* error in GetProcAddress */
            return PCI_E_NAF_NOT_AVAILABLE;

      /* call the "PciPutMessage" entry point of the dll */
      /* and return with the DLL's return code */
      return lpfnPutMessage(ExID, PCIMPB , Message, Data);
      }
```

```
/*
////////////////////////////////////////////////////////////////////////////////
/// PciGetMessage()
*/
PCI_INTEGER far PASCAL PciGetMessage(    PCI_EXID far *ExID,
                              PCI_MPB far *PCIMPB ,
                              PCI_BYTEARRAY Message,
                              PCI_BYTEARRAY Data)
      {
      FARPROC lpfnGetMessage;
      /* get the "PciGetMessage" entry point of the dll */

      lpfnGetMessage = GetProcAddress(ExID->hDLLInstance, "PciGetMessage");
      if (lpfnGetMessage == NULL)   /* error in GetProcAddress */
            return PCI_E_NAF_NOT_AVAILABLE;

      /* call the "PciGetMessage" entry point of the dll */
      /* and return with the DLL's return code */
      return lpfnGetMessage(ExID, PCIMPB , Message, Data);
      }


/*
////////////////////////////////////////////////////////////////////////////////
/// PciSetSignal()
*/
PCI_INTEGER far PASCAL PciSetSignal(     PCI_EXID far *ExID,
                              PCI_INTEGER Signal,
                              PCI_PROCEDURE SignalProcedure)
      {
      FARPROC lpfnSetSignal;

      /* get the "PciSetSignal" entry point of the dll */
      lpfnSetSignal = GetProcAddress(ExID->hDLLInstance, "PciSetSignal");
      if (lpfnSetSignal == NULL)   /* error in GetProcAddress */
            return PCI_E_NAF_NOT_AVAILABLE;

      /* call the "PciSetSignal" entry point of the dll */
      /* and return with the DLL's return code */
      return lpfnSetSignal(ExID, Signal, SignalProcedure);
      }
```

H.3 UNIX Operating System implementation coding samples

These samples present a way to implement the exchange mechanism function call from the PUF point of view.

The **PciGetHandles** function call is not presented.

/* Include files and basic definitions */

```
#include    <stddef.h>
#include    <fcntl.h>
#include    <signal.h>
#include    <stropts.h>
#include    <errno.h>
#include    <stdlib.h>

#define ERROR     (-1)  /* Error value */
#define Success   (0)   /* Success value */

/* Basic types */
typedef int       PCI_INTEGER;
typedef char *          PCI_BYTEARRAY;
typedef int       PCI_EXID;
typedef char *          PCI_HANDLE;
typedef void            (* PCI_PROCEDURE)();

/* Structures */
struct pci_mpb {
      PCI_INTEGER MessageID;
      PCI_INTEGER MessageMaximumSize;
      PCI_INTEGER MessageActualUsedSize;
      PCI_INTEGER DataMaximumSize;
      PCI_INTEGER DataActualUsedSize;
};
```

```
struct pci_register {              /* structure containing registering info */
      PCI_INTEGER PUFVersion;      /* optional: give PUF version */
      PCI_INTEGER PUFType;         /* optional: give PUF type */
      PCI_INTEGER MaxMsgSize; /* return: max size of a message */
};

struct pci_opsys {                       /* structure containing registering info */
      int   DummyParameter;        /* No specific requirement for WINDOWS */
};

/* Function definitions */
#define PCI_PROPERTY               (('Z' << 8) | 1)
#define PCI_REGISTER               (('Z' << 8) | 2)
#define PCI_DEREGISTER             (('Z' << 8) | 3)
#define PCI_SETSIGNAL              (('Z' << 8) | 4)



/*************************************************************
 *    PciGetProperty function
 */
PCI_INTEGER PciGetProperty (PCIHandle, MaximumSize, NAFProperty, ActualSize)
      PCI_HANDLE PCIHandle;              /* char *         */
      PCI_INTEGER MaximumSize;           /* int            */
      PCI_BYTEARRAY NAFProperty          /* char *         */
      PCI_INTEGER * ActualSize;          /* int *          */
{
register int filedes;              /* filedescriptor */
struct strioctl   strioct;         /* stream message control part pointer */

*ActualSize = ERROR;         /* preset with error value */

if ((filedes = open (PCIHandle, O_RDWR)) < Success)
      return ERROR;

strioct.ic_cmd         = PCI_PROPERTY;
strioct.ic_timout            = 0;
strioct.ic_len         = MaximumSize;
strioct.ic_dp          = (char *) NAFProperty;

if (ioctl (filedes, I_STR, &strioct) == 0) {
      *ActualSize = strioct.ic_len;
      close (filedes);
      return 0;
      }
else
      {
      *ActualSize = 0;
      close (filedes);
      return errno;
      }
}

/*************************************************************
 *    PciRegister function
 */
PCI_INTEGER PciRegister (PCIHandle, pciregister, pcidummy, ExID)
      PCI_HANDLE  PCIHandle;  /* char *   */
      struct pci_register     *pciregister;
      struct pci_opsys  *pcidummy;
      PCI_EXID *  ExID;       /* int *    */
{
struct strioctl   strioctl;

struct pci_register_t {
      int         puf_version;
      int         puf_type;
} pci_reg;

pci_reg.puf_version     = pciregister->PUFVersion;
pci_reg.puf_type        = pciregister->PUFType;

strioctl.ic_cmd        = PCI_REGISTER;
strioctl.ic_timout           = 0;
strioctl.ic_len        = sizeof (pci_reg);
strioctl.ic_dp               = (char *) &pci_reg;

if ((*ExID = open (PCIHandle, O_RDWR)) == -1)
      {
      *ExID = 0;
      return errno;
      }
```

```
if ((pciregister->MaxMsgSize = ioctl (*ExID, I_STR, &strioctl)) < 0)
      {
      pciregister->MaxMsgSize = 0;
      close(*ExID);
      *ExID = 0;
      return errno;
      }
else
      {
      return 0;
      }
}



/*****************************************************************
 *    PciDeregister function
 */
PCI_INTEGER PciDeregister (ExID)
      PCI_EXID ExID;    /* int            */
{
struct strioctl   strioctl;

strioctl.ic_cmd         = PCI_DEREGISTER;
strioctl.ic_timout            = 0;
strioctl.ic_len         = 0;
strioctl.ic_dp           = (char *) NULL;

if (ioctl (ExID, I_STR, &strioctl) == -1)
      {
      return errno;
      }
else
      {
      close (ExID);
      return 0;
      }
}
```

```
/*************************************************************
 *    PciPutMessage function
 */
PCI_INTEGER PciPutMessage (ExID, PCIMPB, Message, Data)
      PCI_EXID ExID;               /* int      */
      struct pci_mpb * PCIMPB;
      PCI_BYTEARRAY Message;  /* char *   */
      PCI_BYTEARRAY Data           /* char *   */
{
struct strbuf ctlbuf;
struct strbuf databuf;
char  *buffer = NULL; /* pointer to a buffer, large enough to receive PCIMPB and Message contents
*/
int nErr;

if (Message && ((char *)Message != (char *)PCIMPB + sizeof(struct pci_mpb)))
      {
      /* there is a Message not NULL, and PCIMPB Band Message are not contiguous
       in memory,Have to build a buffer where PCIMPB is followed by the Message content */
      /* Here a memory allocation process may take place */
      buffer = (char *) (malloc( sizeof(struct pci_mpb) + PCIMPB->MessageActualUsedSize));

      memcpy (buffer, PCIMPB, sizeof(struct pci_mpb));
      memcpy (buffer + sizeof(struct pci_mpb), Message, PCIMPB->MessageActualUsedSize);
      ctlbuf.buf  = buffer;
      ctlbuf.len  = PCIMPB->MessageActualUsedSize + sizeof(struct pci_mpb);
}
else
      {
      /* either there is no Message, or the PCIMPB and the Message are contiguous in memory */
      ctlbuf.buf  = (char *)PCIMPB;
      ctlbuf.len  =   Message   ?   PCIMPB->MessageActualUsedSize   +   sizeof(struct   pci_mpb):
sizeof(struct pci_mpb);
      }

databuf.buf = Data;
databuf.len = Data ? PCIMPB->DataActualUsedSize: 0;

if (putmsg (ExID, &ctlbuf, &databuf, 0) != 0)
      nErr = errno;     /* errno contents the error code */
      }
else
      {
      nErr = 0;
      }
if (buffer != NULL) free(buffer);
return nErr;
}
```

```
/*************************************************************
 *    PciGetMessage function
 */
PCI_INTEGER PciGetMessage (ExID, PCIMPB, Message, Data)
      PCI_EXID ExID;               /* int      */
      struct pci_mpb * PCIMPB;
      PCI_BYTEARRAY Message;       /* char *   */
      PCI_BYTEARRAY Data;          /* char *   */
{
struct strbuf ctlbuf;
int flags;
struct strbuf databuf;
char   *buffer = NULL; /* pointer to a buffer,large enough to receive PCIMPB and  the Message
content */
int nErr = 0;

if (Message && ((char *)Message != (char *)PCIMPB + sizeof(struct pci_mpb)))
      {
      /* there is a Message not NULL and,  PCIMPB and Message are not contiguous in memory,
      have to reserve a buffer where PCIMPB can be followed by the Message content */
      /* Here a memory allocation process may take place */
      buffer = (char *) (malloc( sizeof(struct pci_mpb) + PCIMPB->MessageMaximumSize ));
      ctlbuf.buf  = buffer;
      }
else {
      /* either there is no Message, or the PCIMPB and the Message are contiguous in memory */
      ctlbuf.buf  =  (char *)PCIMPB;
      }
ctlbuf.maxlen    = Message ? PCIMPB->MessageMaximumSize + sizeof(struct pci_mpb):sizeof(struct
pci_mpb);
databuf.buf = Data;
databuf.maxlen   = Data ? PCIMPB->DataMaximumSize: 0;

flags = 0;
if (getmsg (ExID, &ctlbuf, &databuf, &flags) != 0)
      {
      /* Error condition, errno will be set    */
      nErr = errno;
}else {
      /* Operation OK   */
      if (ctlbuf.len != -1 && ctlbuf.len >= sizeof(struct pci_mpb)) {
            /* Message, possibly of size 0 is present */
            PCIMPB->MessageActualUsedSize = ctlbuf.len - sizeof(struct pci_mpb);
            if (Message && ((char *)Message != (char *)PCIMPB +
                                    sizeof(struct pci_mpb)))
                  {
                  /* there is a Message not NULL and, PCIMPB and Message are not
                     contiguous in memory, a buffer where PCIMPB can be followed
                     by the Message content, has been used */
                  memcpy ( PCIMPB, buffer, sizeof(struct pci_mpb));
                  memcpy ( Message,(buffer + sizeof(struct pci_mpb)), (ctlbuf.len - sizeof(struct
pci_mpb)));
                  }
            else
                  {
                  /* PCIMPB and Message are contiguous in memory, no more buffer used */
                  Message = (char *) (PCIMPB + sizeof(struct pci_mpb));
                  }
      }else {
            /* No Message present or too small message: error at least PCIMPB
               should be there */
            PCIMPB->MessageID = 0;
            PCIMPB->MessageActualUsedSize = 0;
            }


if (databuf.len != -1)
      {
      /* Data block, possibly of size 0 is present */
      PCIMPB->DataActualUsedSize    = databuf.len;
      }
else
      {
      /* No Data present */
      PCIMPB->DataActualUsedSize = 0;
      }
}
if (buffer != NULL) free( buffer);

return nErr;
}
```

```
/*****************************************************************
 *    PciSetSignal function
 */
PCI_INTEGER PciSetSignal (ExID, Signal, SignalProcedure)

PCI_EXID ExID;               /* int            */
PCI_INTEGER Signal;          /* int            */
PCI_PROCEDURE SignalProcedure;     /* void (*) ()    */
{
if (Signal == 0)
      {
      if (ioctl (ExID, I_SETSIG, 0) == -1)
            return errno;
      signal (SIGPOLL, SIG_DFL);
      return 0;
      }
else
      {
      if (ioctl (ExID, I_SETSIG, S_MSG) == -1)
            return errno;
      signal (SIGPOLL, SignalProcedure);
      return 0;
      }
}
```

H.4 OS/2 Operating System implementation coding samples

H.4.1 Sample OS/2 application level implementation coding

The following code shows a sample implementation of PUF exchange functions for the OS/2 application level environment. The sample is illustrated using "C" language:

```
/*
 * standard includes
 */
#define INCL_DOSSEMAPHORES
#define INCL_DOSMEMMGR
#define INCL_DOS
#define INCL_SUB
#include <os2.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>

/*
 * Basic types
 */
typedef SHORT   PCI_INTEGER;
typedef PSZ     PCI_BYTEARRAY;
typedef PSZ     PCI_HANDLE;
typedef struct  {
      HMODULE hDLLInstance;
      PCI_INTEGER     Exchange_Id;
      } PCI_EXID;

typedef ULONG *   PCI_PROCEDURE;

/*
 * PCI Structures
 */
struct pci_mpb {
            PCI_INTEGER     MessageID;
            PCI_INTEGER     MessageMaximumSize;
            PCI_INTEGER     MessageActualUsedSize;
            PCI_INTEGER     DataMaximumSize;
            PCI_INTEGER     DataActualUsedSize;
};
typedef struct pci_mpb PCI_MPB;

struct pci_register {              /* structure containing registering info */
      PCI_INTEGER PUFVersion; /* optional: give PUF version */
      PCI_INTEGER PUFType;         /* optional: give PUF type */
      PCI_INTEGER MaxMsgSize; /* return: max size of a message */
};


struct pci_opsys {                 /* structure containing specific operating system info */
      int         DummyParameter;   /* No specific requirement for OS2 */
};
```

```
/*
 * PCI defines
 */
#define PCI_HANDLE_LENGTH    254 /*size of each handle in the buffer from PciGetHandles */
#define PCI_E_SUCCESS                    0
#define PCI_E_QUERY_ENTITY_NOT_AVAILABLE 128
#define PCI_E_INVALID_PCI_HANDLE         130
#define PCI_E_NAF_NOT_AVAILABLE          255

/*
////////////////////////////////////////////////////////
/// PciGetHandles()
*/
PCI_INTEGER far PASCAL PciGetHandles ( PCI_INTEGER MaxHandles,
                  PCI_HANDLE PCIHandles,
                  PCI_INTEGER far * ActualHandles)
    {
    int     nafNumber;
    int     nafFound;
    int     size;
    char    keyName[20];
    PCI_BYTEARRAY buffer;
    HINI    hini;
    hini = PrfOpenProfile(hini, "PCI.INI);
    buffer = PCIHandles;
    for (nafNumber = 1, nafFound = 0; nafNumber <= MaxHandles;nafNumber++)
      {
#ifdef PCI_DLL                      /* For OS/2 application level */
sprintf( keyName, "pciDLL%d", index);
size = PrfQueryProfileString(hini,
           "PCI_DLL",  /* Section name */
#endif
#ifdef PCI_PDD                      /* For OS/2 device driver level */
sprintf( keyName, "pciDriver%d", index);
size = PrfQueryProfileString(hini,
           "PCI_PDD",  /* Section name */
#endif
                  keyName,          /* 'pciDriver'+1..n */
                  NULL,             /* No default string needed */
                  buffer,                /* Address  where to put the result */
                  PCI_HANDLE_LENGTH);    /* Maxi. size for the result */
      if (size > 0)
         {
           nafFound++;   /* One more NAF found */
           buffer  +=  PCI_HANDLE_LENGTH;/* Next location for  a  PCIHandle */
         }
      }
    *ActualHandles = nafFound;

    PrfCloseProfile(hini);
    /* return with the DLL's return code */
    return PCI_E_SUCCESS;
    }
```

```
/*
////////////////////////////////////////////////////////////
/// PciGetProperty()
*/
PCI_INTEGER far PASCAL PciGetProperty (PCI_HANDLE PCIHandle,
                 PCI_INTEGER MaximumSize,
                 PCI_BYTEARRAY Property,
                 PCI_INTEGER far * ActualSize)
    {
    PCI_INTEGER iReturnCode;
    HMODULE hDLLInstance;
    BYTE Failname[256];
    PCI_INTEGER Failsize;
    PFN lpfnGetProperty;

    /* load the NAF's DLL and get the instance DLL */
    if ( DosLoadModule( Failname, Failsize, PCIHandle, &hDLLInstance) !=0)
        return   PCI_E_INVALID_PCI_HANDLE;    /* error in Load DLL */

    /* get the "PciGetProperty" entry point of the dll */
    DosQueryProcAddr(hDLLInstance, 1,"PciGetProperty", &lpfnGetProperty);
    if (lpfnGetProperty == NULL)
      {
        DosFreeModule(hDLLInstance);
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */
      }

    /* call the "PciGetProperty" entry point of the dll */
    iReturnCode = lpfnGetProperty(PCIHandle,  MaximumSize, Property, ActualSize);

    /* free the DLL in any case */
    DosFreeModule(hDLLInstance);

    /* return with the DLL's return code */
    return iReturnCode;
    }

/*
////////////////////////////////////////////////////////////
/// PciRegister()
/// The PCIOpSysInfo is kept for compatibility only
*/
PCI_INTEGER far PASCAL PciRegister (   PCI_HANDLE PCIHandle,
                 struct pci_register *PCIRegisterInfo,
                 struct pci_opsys * PCIOpSysInfo,
                 PCI_EXID far *ExID)
    {
    PCI_INTEGER iReturnCode;
    PFN lpfnRegister;
    HMODULE hDLLInstance;
    BYTE Failname[256];
    PCI_INTEGER Failsize;


    /* load the NAF's DLL and get the instance DLL */
    if ( DosLoadModule( Failname, Failsize, PCIHandle, &hDLLInstance) !=0)
        return   PCI_E_INVALID_PCI_HANDLE;    /* error in Load DLL */

    /* put the DLL instance in ExID */
    ExID->hDLLInstance = hDLLInstance;

    /* get the "PciRegister" entry point of the dll */
    DosQueryProcAddr(hDLLInstance, 2,"PciRegister", &lpfnRegister);
    if (lpfnRegister == NULL)
      {
        DosFreeModule(hDLLInstance);
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */
      }

    /* call the "PciRegister" entry point of the dll */
    iReturnCode = lpfnRegister(PCIRegisterInfo, ExID);

    if (iReturnCode != 0)
      {   /* error in PciRegister: free the DLL */
        DosFreeModule(hDLLInstance);
      }

    /* return with the DLL's return code */
    return iReturnCode;
    }
```

```
/*
////////////////////////////////////////////////////////////
/// PciDeRegister()
*/
PCI_INTEGER far PASCAL PciDeregister(PCI_EXID far *ExID)
    {
    PCI_INTEGER iReturnCode;
    PFN lpfnDeregister;

    /* get the "PciDeregister" entry point of the dll */
    DosQueryProcAddr(ExID->hDLLInstance, 3,"PciDeregister", &lpfnDeregister);
    if (lpfnDeregister == NULL)
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */


    /* call the "PciDeRegister" entry point of the dll */
    iReturnCode = lpfnDeregister(ExID);

    /* free the DLL in any case */
    DosFreeModule(ExID->hDLLInstance);

    /* return with the DLL's return code */
    return iReturnCode;
    }

/*
////////////////////////////////////////////////////////////
/// PciPutMessage()
*/
PCI_INTEGER far PASCAL PciPutMessage(  PCI_EXID far *ExID,
                  PCI_MPB far *PCIMPB ,
                  PCI_BYTEARRAY Message,
                  PCI_BYTEARRAY Data)
    {
    PFN lpfnPutMessage;

    /* get the "PciPutMessage" entry point of the dll */
    DosQueryProcAddr(ExID->hDLLInstance, 4,"PciPutMessage", &lpfnPutMessage);
    if (lpfnPutMessage == NULL)
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */


    /* call the "PciPutMessage" entry point of the dll */
    /* and return with the DLL's return code */

    return lpfnPutMessage(ExID, PCIMPB , Message, Data);
    }

/*
////////////////////////////////////////////////////////////
/// PciGetMessage()
*/
PCI_INTEGER far PASCAL PciGetMessage(  PCI_EXID far *ExID,
                  PCI_MPB far *PCIMPB ,
                  PCI_BYTEARRAY Message,
                  PCI_BYTEARRAY Data)
    {
    PFN lpfnGetMessage;

    /* get the "PciGetMessage" entry point of the dll */
    DosQueryProcAddr(ExID->hDLLInstance, 5,"PciGetMessage", &lpfnGetMessage);
    if (lpfnGetMessage == NULL)
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */

    /* call the "PciGetMessage" entry point of the dll */
    /* and return with the DLL's return code */

    return lpfnGetMessage(ExID, PCIMPB , Message, Data);
    }
```

```
/*
////////////////////////////////////////////////////////////
/// PciSetSignal()
*/
PCI_INTEGER far PASCAL PciSetSignal(   PCI_EXID far *ExID,
                 PCI_INTEGER Signal,
                 PCI_PROCEDURE SignalProcedure)
    {
    PFN lpfnSetSignal;

    /* get the "PciSetSignal" entry point of the dll */
    DosQueryProcAddr(ExID->hDLLInstance, 6, "PciSetSignal", &lpfnSetSignal);
    if (lpfnSetSignal == NULL)
        return   PCI_E_NAF_NOT_AVAILABLE;   /* error in GetProcAddress */

    /* call the "PciSetSignal" entry point of the dll */
    /* and return with the DLL's return code */
    return lpfnSetSignal(ExID, Signal, SignalProcedure);
    }
```

H.4.2 Sample OS/2 device driver level implementation coding

The following code shows a sample implementation of PUF exchange functions for the OS/2 device driver level environment. The sample is illustrated using "C" language:

```
/*
 * Include files
 */
#include <os2.h>
#include <dos.h>
#include <fcntl.h>
#include <memory.h>
#include <malloc.h>
#include <stdio.h>

/*
 * General typedefs
 */
typedef void            ( *   PFRV) ();     /* Pointer to Function Returning Void   */
typedef short int       ( far *  FPFRS) (); /* Far Pointer to Function Returning Short      */
typedef void            ( far *  FPFRV) (); /* Far Pointer to Function Returning Void       */
typedef int             ( far *  FPFRI) (); /* Far Pointer to Function Returning Int        */

/*
 * Mapping of generic type definitions
 */

typedef short int               PCI_INTEGER;
typedef char far        *       PCI_BYTEARRAY;
typedef struct {
      short int                 (far *hNaf)();
      PCI_INTEGER     Exchange_Id;
      unsigned short  Naf_Ds;
      } PCI_EXID;
typedef char far        *       PCI_HANDLE;
typedef FPFRV                   PCI_PROCEDURE;

/*
* Definition of function codes
*/

#define PCIGETPROPERTY  (short) (1)
#define PCIREGISTER     (short) (2)
#define PCIDEREGISTER   (short) (3)
#define PCIPUTMESSAGE   (short) (4)
#define PCIGETMESSAGE   (short) (5)
#define PCISETSIGNAL    (short) (6)

/*
 * Error definitions
 */

#define E_DEVICE_DRIVER_NOT_FOUND        128
#define E_DEVICE_DRIVER_CONTROL          128
#define E_NAF_NOT_FOUND             130
#define E_NAF_INVALID_ADDRESS            130
#define E_TOO_MANY_ASSOCIATIONS          133
#define E_INVALID_EXCHANGE_ID            136

/*
 * Other definitions
 */
```

```
#define SUCCESS                         0
#define MAX_EXID                        32 /* allow 32 PUF_NAF associations */

/*
 * Structures
 */

struct pci_mpb {
      PCI_INTEGER MessageID;
      PCI_INTEGER MessageMaximumSize;
      PCI_INTEGER MessageActualUsedSize;
      PCI_INTEGER DataMaximumSize;
      PCI_INTEGER DataActualUsedSize;
};

struct pci_register {                   /* structure containing registering info */
      PCI_INTEGER PUFVersion;         /* optional: give PUF version */
      PCI_INTEGER PUFType;            /* optional: give PUF type */
      PCI_INTEGER MaxMsgSize;         /* return: max size of a message */
};

struct pci_opsys {                              /* structure containing registering info */
      short int               MaxNCOCount;     /* optional: give max count of NCOs */
      short int               MaxPacketSize;   /* optional: give expected max size and */
      short int               MaxPacketCount;  /* max count of packets to buffer */
      long int                AddBufferSize;   /* optional: give to NAF size and */
      void far *              AddBufferSpace;  /* pointer to additional buffer */
      long int                BufferNeeded;    /* return: amount of add buffer needed */
};

typedef struct  {                    /* structure containing IDC entry point info */
      unsigned short RealOffset;    /* real mode offset of IDC entry point  */
      unsigned short RealSegment;   /* real mode segment of IDC entry point */
      unsigned short RealDs;        /* real mode DS of IDC entry point */
      unsigned short ProtOffset;    /* protect mode offset of IDC entry point */
      unsigned short ProtSegment;   /* protect mode segment of IDC entry point */
      unsigned short ProtDs;        /* protect mode DS of IDC entry point   */
}PCI_ATTACHAREA;

/*
* Functional constants
*/
const char PCIsign[8]="ISDN PCI";

/*
////////////////////////////////////////////////////////////
/// PciGetHandles()
*/
PCI_INTEGER PciGetHandles ( PCI_INTEGER MaxHandles,
                  PCI_HANDLE PCIHandles,
                  PCI_INTEGER far * ActualHandles)
    {
    int               nafNumber;
    int           nafFound;
    int               size;
    char          keyName[20];
    PCI_BYTEARRAY buffer;
    HINI          hini;

    hini = PrfOpenProfile(hini, "PCI.INI);
    buffer = PCIHandles;
    for (nafNumber = 1, nafFound = 0; nafNumber <= MaxHandles;nafNumber++)
      {
       #ifdef PCI_DLL                    /* For OS/2 application level */
       sprintf( keyName, "pciDLL%d", index);
       size = PrfQueryProfileString(hini,
                  "PCI_DLL",  /* Section name */
       #endif
       #ifdef PCI_PDD                    /* For OS/2 device driver level */
       sprintf( keyName, "pciDriver%d", index);
       size = PrfQueryProfileString(hini,
                  "PCI_PDD",  /* Section name */
       #endif
                  keyName, /* 'pciDriver'+1..n */
                  NULL,     /*        No  default string needed */
                  buffer,        /* Address    where to put the result */
                  PCI_HANDLE_LENGTH);      /*  Maxi.  size   for   the result */
      if (size > 0)
         {
           nafFound++;   /* One more NAF found */
           buffer  +=  PCI_HANDLE_LENGTH;/* Next location for  a  PCIHandle */
         }
```

```
      }
    *ActualHandles = nafFound;
    PrfCloseProfile(hini);
    /* return with the DLL's return code */
    return SUCCESS;
    }
/*
////////////////////////////////////////////////////////////
PciGetProperty:        Asks the NAF for it's properties, which is a list of TLV coded topics.
      Returns the properties into the given Property buffer.
      The maximum size of the Property buffer is given in MaximumSize.
      Function will fail, if MaximumSize is less than the size of the Property the
      NAF can deliver.
*/
short int PciGetProperty ( PCI_HANDLE PCIHandle,
      short int MaximumSize,
      char * Property,
      short int * ActualSize )
{
register short int error;
PCI_ATTACHAREA AttachArea;
short int  (far *hNAF)();

/* Gain NAF PDD access */
if ( AttachDD("DEVICE  ",&AttachArea))
      return E_NAF_INVALID_ADDRESS;   /* NAF inaccessible, invalid address */
else
    (void far *) hNAF = MAKEP(AttachArea.ProtSegment, AttachArea.ProtOffset);

/* Check if NAF is available */
if ( hNAF == NULL)
      return E_NAF_INVALID_ADDRESS;   /* NAF inaccessible, invalid address */

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciGetProperty( AttachArea.ProtDs,
              hNAF,
              PCIGETPROPERTY,
              MaximumSize,
              (char far *) Property,
              (short int far *) ActualSize );

return error;
}       /* End of PciGetProperty() */

/*
////////////////////////////////////////////////////////////
PciRegister:   Tries to associate calling PUF with selected NAF.
      Delivers the ExID, which has to be used in subsequent calls.
      Two structures have to be provided by the calling PUF:
      -       The PCIRegisterInfo and
      -       the PCIOpSysInfo structure.
*/
short int PciRegister ( PCI_HANDLE PCIHandle,
      struct pci_register * PCIRegisterInfo,
      struct pci_opsys * PCIOpSysInfo,
      PCI_EXID * ExID )
{
register short int error;
PCI_ATTACHAREA AttachArea;
short int  (far *hNAF)();

/* Gain NAF PDD access */
if ( AttachDD("DEVICE  ",&AttachArea))
      return E_NAF_INVALID_ADDRESS;   /* NAF inaccessible, invalid address */
else
    (void far *) hNAF = MAKEP(AttachArea.ProtSegment, AttachArea.ProtOffset);

/* Check if NAF is available */
if ( hNAF == NULL)
      return E_NAF_INVALID_ADDRESS;   /* NAF inaccessible, invalid address */

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciRegister( AttachArea.ProtDs,
            hNAF,
            PCIREGISTER,
            (struct pci_register far *) PCIRegisterInfo,
            (struct pci_opsys far *) PCIOpSysInfo,
            (short int far * ) ExID );
```

```
if ( ! error)
      {
      /* Association was successful */
      ExID->hNaf = hNAF;
      ExID->Naf_Ds = AttachArea.ProtDs;
      /* Save our Ds with an assembly routine */
      SaveOurDs();
      }

return error;
}       /* End of PciRegister() */

/*
////////////////////////////////////////////////////////////
PciDeregister: Terminates an existing association wit a NAF.
      The ExID of an existing association has to be provided.
*/

short int PciDeregister (  PCI_EXID *ExID )
{
register short int error;

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciDeregister( ExID->Naf_Ds,
            ExID->hNaf,
            PCIDEREGISTER,
            ExID->Exchange_Id);

return error;
}       /* End of PciDeregister() */


/*
////////////////////////////////////////////////////////////
PciPutMessage: Transfers a Message and associated Data to the NAF.
*/

short int PciPutMessage (  PCI_EXID * ExID,
      struct pci_mpb * PCIMPB ,
      char * Message,
      char * Data )
{
register short int error;

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciPutMessage( ExID->Naf_Ds,
            ExID->hNaf,
            PCIPUTMESSAGE,
            ExID->Exchange_Id,
            (struct pci_mpb far *) PCIMPB ,
            (char far *) Message,
            (char far *) Data );
return error;
}       /* End of PciPutMessage() */
```

```
/*
////////////////////////////////////////////////////////////
PciGetMessage: Receives a Message and associated Data from the NAF.
*/

short int PciGetMessage (  PCI_EXID * ExID,
      struct pci_mpb * PCIMPB ,
      char far * Message,
      char far * Data )
{
register short int error;

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciGetMessage( ExID->Naf_Ds,
            ExID->hNaf,
            PCIGETMESSAGE,
            ExID->Exchange_Id,
            (struct pci_mpb far *) PCIMPB ,
            (char far *) Message,
            (char far *) Data );

return error;
}       /* End of PciGetMessage() */

/*
////////////////////////////////////////////////////////////
PciSetSignal: Hands the address of a SignalProcedure to the NAF.
      The SignalProcedure then will receive notification on communication
      events (i.e. Message available for retrieval)
*/

short int PciSetSignal (   PCI_EXID * ExID,
      short int Signal,
      FPFRV SignalProcedure )
{
register short int error;

/* Call the NAF IDC entry point with an assembly routine to set-up DS */
error = CallIDCPciSetSignal( ExID->Naf_Ds,
            ExID->hNaf,
            PCISETSIGNAL,
            ExID->Exchange_Id,
            (FPFRV) SignalProcedure );

return error;
}       /* End of PciSetSignal() */
```

```
/*
;
; Assembly routines
;

_TEXT      SEGMENT
      ASSUME  CS: _TEXT

our_ds          dw      ?

            PUBLIC  _SaveOurDs
_SaveOurDs      PROC NEAR
            push    ax
            mov     ax,ds
            mov     cs:our_ds,ax
            pop     ax
            ret
_SaveOurDs      ENDP
;
;
;
            PUBLIC  _CallIDCGetProperty
_CallIDCGetProperty     PROC NEAR
            push    bp
            mov     bp,sp
;
; In the stack
;
;                   AttachArea.ProtDs                   20
;                   hNAF,                               16
;                   PCIGETPROPERTY                      14
;                   MaximumSize                         12
;                   (char far *) Property               8
;                   (short int far *) ActualSize )      4
;
; Push parameters
;
            push     WORD PTR [bp+4]
            push     WORD PTR [bp+6]
            push     WORD PTR [bp+8]
            push     WORD PTR [bp+10]
            push     WORD PTR [bp+12]
            push     WORD PTR [bp+14]
;
; Set-up NAF's DS
;
            mov     ax , WORD PTR [bp+20]
            mov     ds,ax
;
; call IDC entry point
;
            call    DWORD PTR [bp+16]        ;hNAF
            add     sp, 12
```

```
;
; Restore our DS
;
            push    ax      ;save return code
            mov     ax,cs:our_ds
            mov     ds,ax
            pop     ax      ;restore return code

;
; Return to C calling function
;
            pop     bp
            ret

_CallIDCGetProperty   ENDP

            PUBLIC  _CallIDCPciRegister
_CallIDCPciRegister     PROC NEAR

;
;       same mechanism ...
;

_CallIDCPciRegister     ENDP

;
;       etc...
;

_TEXT   ENDS
END
*/
```

H.5 Sample Windows NT implementation coding samples

H.5.1 User mode PUF / User mode NAF

The following code shows a sample DLL implementation of **PUF** exchange functions for the Windows NT environment. The sample is illustrated using "C" language:

```
// Standard includes
#include <windows.h>
#include "puf.h"

//
// PciGetHandles()
//
PCI_INTEGER PASCAL PciGetHandles  ( PCI_INTEGER MaxHandles,
                                    PCI_HANDLE PCIHandles,
                                    PCI_INTEGER  * ActualHandles )
{
int               nafNumber;
int               nafFound;
char              keyName[20];
PCI_BYTEARRAY     buffer;
INT               i;

HKEY              hKey;
CHAR              ValueName[100];
DWORD             cbValueName;
DWORD             dwType;
DWORD             retCode;

CHAR              ClassName[200];
DWORD             dwcClassLen = 200;
DWORD             dwcSubKeys;
DWORD             dwcMaxSubKey;
DWORD             dwcMaxClass;
DWORD             dwcValues;
DWORD             dwcMaxValueName;
DWORD             dwcMaxValueData;
DWORD             dwcSecDesc;
FILETIME          ftLastWriteTime;
DWORD             cbData;

buffer = PCIHandles;

// Open the PCI\Drivers Key of the registry
retCode = RegOpenKeyEx (HKEY_LOCAL_MACHINE,
                        "SOFTWARE\\PCI\\DRIVERS",
                        0,
                        KEY_EXECUTE,
                        &hKey);

if (retCode) return 0;
// ADD A QUERY AND ALLOCATE A BUFFER FOR BDATA.
retCode = RegQueryInfoKey(hKey,              // Key handle.
                          ClassName,         // Buffer for class name.
                          &dwcClassLen,      // Length of class string.
                          NULL,              // Reserved.
                          &dwcSubKeys,       // Number of sub keys.
                          &dwcMaxSubKey,     // Longest sub key size.
                          &dwcMaxClass,      // Longest class string.
                          &dwcValues,        // Number of values for this key.
                          &dwcMaxValueName,  // Longest Value name.
                          &dwcMaxValueData,  // Longest Value data.
                          &dwcSecDesc,       // Security descriptor.
                          &ftLastWriteTime); // Last write time.
if (retCode) return 0;
cbData = 128;

// ENUMERATE THE KEY.
for (nafNumber=1, nafFound=0, i=0;
    (nafNumber <= MaxHandles) && (retCode != ERROR_NO_MORE_ITEMS) ;
     nafNumber++ , i++) {
  cbValueName = 100;
  retCode = RegEnumValue (hKey,              // Key handle returned from RegOpenKeyEx.
                          i,                 // Index, taken from listbox
                          ValueName,         // Name of value.
                          &cbValueName,      // Size of value name.
                          NULL,              // Reserved, dword = NULL.
                          &dwType,           // Type of data.
                          buffer,            // Data buffer.
                          &cbData);          // Size of data buffer.
  wsprintf( keyName , "pciDriver%d" , nafNumber );
```

```
  if( (cbData > 0) && (stricmp( ValueName , keyName ) == 0) ) {
    nafFound++;
    buffer += 128; // Next location for a PCIHandle (128 octets fixed size).
  }
}

// Close the PCI\Drivers Key of the registry
RegCloseKey(hKey);

*ActualHandles = nafFound;
return PCI_E_SUCCESS;
}

//
// PciGetProperty()
//
PCI_INTEGER  PASCAL PciGetProperty ( PCI_HANDLE PCIHandle,
                                     PCI_INTEGER MaximumSize,
                                     PCI_BYTEARRAY Property,
                                     PCI_INTEGER *ActualSize)
{
PCI_INTEGER       iReturnCode;
HINSTANCE         hDLLInstance;
FARPROC           lpfnGetProperty;

// Load the NAF's DLL and return error if failed
hDLLInstance = LoadLibrary(PCIHandle);
if( hDLLInstance < HINSTANCE_ERROR ) return PCI_E_INVALID_PCI_HANDLE;

// Get the "PciGetProperty" entry point of the dll.
lpfnGetProperty = GetProcAddress( hDLLInstance , "PciGetProperty" );
if( lpfnGetProperty == NULL ) {
  FreeLibrary( hDLLInstance );
  return PCI_E_NAF_NOT_AVAILABLE;  // Error in GetProcAddress.
}

// Call the "PciGetProperty" entry point of the dll.
iReturnCode = lpfnGetProperty(PCIHandle,MaximumSize,Property,ActualSize);
// Freelibrary in any case.
FreeLibrary( hDLLInstance );
return iReturnCode;
}

//
// PciRegister()
//
PCI_INTEGER  PASCAL PciRegister ( PCI_HANDLE PCIHandle,
                                  struct pci_register *PCIRegisterInfo,
                                  struct pci_opsys * PCIOpSysInfo,
                                  PCI_EXID  *ExID )
{
PCI_INTEGER       iReturnCode;
HINSTANCE         hDLLInstance;
FARPROC           lpfnRegister;

// Load the NAF's DLL and return error if failed
hDLLInstance = LoadLibrary(PCIHandle);
if( hDLLInstance < HINSTANCE_ERROR ) return PCI_E_INVALID_PCI_HANDLE;
ExID->hDLLInstance = hDLLInstance;

// Get the "PciRegister" entry point of the dll.
lpfnRegister = GetProcAddress( hDLLInstance , "PciRegister" );
if( lpfnRegister == NULL ) {
  FreeLibrary( hDLLInstance );
  return PCI_E_NAF_NOT_AVAILABLE;  // Error in GetProcAddress.
}
// Call the "PciRegister" entry point of the dll.
iReturnCode = lpfnRegister(PCIRegisterInfo, ExID);
if( iReturnCode != 0 ) FreeLibrary( hDLLInstance );
return iReturnCode;
}
```

```
//
// PciDeRegister()
//
PCI_INTEGER  PASCAL PciDeregister ( PCI_EXID  *ExID )
{
PCI_INTEGER       iReturnCode;
FARPROC           lpfnDeRegister;

// Get the "PciDeRegister" entry point of the dll, return error if failed
lpfnDeRegister = GetProcAddress( ExID->hDLLInstance , "PciDeRegister" );
if( lpfnDeRegister == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
ExID->hDLLInstance = 0;

// Call the "Pci" entry point of the dll.
iReturnCode = lpfnDeRegister( ExID );

// FreeLibrary in any case.
FreeLibrary( ExID->hDLLInstance );
return iReturnCode;
}

//
// PciPutMessage()
//
PCI_INTEGER  PASCAL PciPutMessage ( PCI_EXID  *ExID,
                                    PCI_MPB  *PCIMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data )
{
FARPROC           lpfnPutMessage;

// Get the "PciPutMessage" entry point of the dll.
lpfnPutMessage = GetProcAddress( ExID->hDLLInstance , "PciPutMessage" );
if( lpfnPutMessage == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
return lpfnPutMessage(ExID,PCIMPB,Message,Data);
}

//
// PciGetMessage()
//
PCI_INTEGER  PASCAL PciGetMessage ( PCI_EXID  *ExID,
                                    PCI_MPB  *PCIMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data )
{
FARPROC  lpfnGetMessage;

// Get the "PciGetMessage" entry point of the dll.
lpfnGetMessage = GetProcAddress( ExID->hDLLInstance , "PciGetMessage" );
if( lpfnGetMessage == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
return lpfnGetMessage(ExID,PCIMPB,Message,Data);
}

//
// PciSetSignal()
//
PCI_INTEGER  PASCAL PciSetSignal ( PCI_EXID  *ExID,
                                   PCI_INTEGER Signal,
                                   PCI_PROCEDURE SignalProcedure)
{
FARPROC  lpfnSetSignal;

// Get the "PciSetSignal" entry point of the dll.
lpfnSetSignal = GetProcAddress( ExID->hDLLInstance , "PciSetSignal" );
if( lpfnSetSignal == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
return lpfnSetSignal(ExID,Signal,SignalProcedure);
}
```

H.5.2 User mode PUF / Kernel mode NAF

In this subclause, the software architecture is illustrated in figure H.1. Only a subset of functions preseted which are: PciGetHandles, PciRegister, PciDeregister, PciGetMessage.

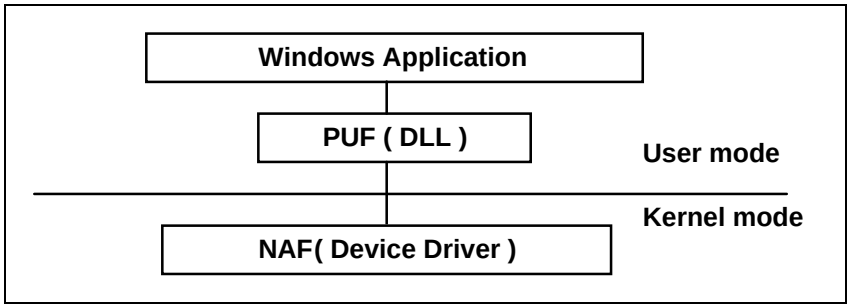


Figure H.1: User mode PUF sample

```
//
// PciGetHandles()
//
PCI_INTEGER PASCAL PciGetHandles(   PCI_INTEGER MaxHandles,
                                    PCI_HANDLE PCIHandles,
                                    PCI_INTEGER  * ActualHandles )
{
int               nafNumber;
int               nafFound;
char              keyName[20];
PCI_BYTEARRAY     buffer;
INT               i;

HKEY              hKey;
CHAR              ValueName[100];
DWORD             cbValueName;
DWORD             dwType;
DWORD             retCode;

CHAR              ClassName[200];
DWORD             dwcClassLen = 200;
DWORD             dwcSubKeys;
DWORD             dwcMaxSubKey;
DWORD             dwcMaxClass;
DWORD             dwcValues;
DWORD             dwcMaxValueName;
DWORD             dwcMaxValueData;
DWORD             dwcSecDesc;
FILETIME          ftLastWriteTime;


DWORD  cbData;

         buffer = PCIHandles;

// Open the PCI\Drivers Key of the registry
retCode = RegOpenKeyEx (HKEY_LOCAL_MACHINE,
                        "SOFTWARE\\PCI\\DEVICEDRIVERS",
                        0,
                        KEY_EXECUTE,
                        &hKey);
if (retCode)
         return 0;
// ADD A QUERY AND ALLOCATE A BUFFER FOR BDATA.
retCode =RegQueryInfoKey (          hKey,               // Key handle.
                                    ClassName,          // Buffer for class name.
                                    &dwcClassLen,       // Length of class string.
                                    NULL,               // Reserved.
                                    &dwcSubKeys,        // Number of sub keys.
                                    &dwcMaxSubKey,      // Longest sub key size.
                                    &dwcMaxClass,       // Longest class string.
                                    &dwcValues,         // Number of values for this key.
                                    &dwcMaxValueName,   // Longest Value name.
                                    &dwcMaxValueData,   // Longest Value data.
                                    &dwcSecDesc,        // Security descriptor.
                                    &ftLastWriteTime); // Last write time.
```

```
if (retCode)
        return 0;

cbData = 128;

// ENUMERATE THE KEY.
for( nafNumber = 1 , nafFound = 0 , i = 0;
         (nafNumber <= MaxHandles) && (retCode != ERROR_NO_MORE_ITEMS) ;
         nafNumber++ , i++)
        {
        cbValueName = 100;
        retCode = RegEnumValue (   hKey,             // Key handle returned from RegOpenKeyEx
                                   i,                // Value index, taken from listbox.
                                   ValueName,        // Name of value.
                                   &cbValueName,     // Size of value name.
                                   NULL,             // Reserved, dword = NULL.
                                   &dwType,          // Type of data.
                                   buffer,           // Data buffer.
                                   &cbData);         // Size of data buffer.

        wsprintf( keyName , "pciDeviceDriver%d" , nafNumber );
        if( (cbData > 0) && (stricmp( ValueName , keyName ) == 0) )
                {
                nafFound++;
                buffer += 128;    // Next location for a PCIHandle ( 128   octets fixed size)
                }
        }

// Close the PCI\Drivers Key of the registry
RegCloseKey( hKey );

*ActualHandles = nafFound;

return PCI_E_SUCCESS;
}

//
// PciRegister()
//
PCI_INTEGER  PASCAL PciRegister ( PCI_HANDLE PCIHandle,
                                     struct pci_register *PCIRegisterInfo,
                                     struct pci_opsys * PCIOpSysInfo,
                                     PCI_EXID  *ExID )
{
PCI_INTEGER      iReturnCode;
struct IoPciRegister pR;
BOOL             RetDevIo;
DWORD            BR;

// Open device driver.
ExID->hDLLInstance = CreateFile(    PCIHandle,
                                   GENERIC_READ,
                                   FILE_SHARE_READ,
                                   NULL,
                                   OPEN_EXISTING,
                                   0,
                                   NULL );

if( ExID->hDLLInstance != INVALID_HANDLE_VALUE )
        {
        // Copy parameters to the input buffer.
        pR.ExID = *ExID;
        pR.PciRegisterInfo = *PCIRegisterInfo;

        // Call device driver.
        RetDevIo = DeviceIoControl(        ExID->hDLLInstance,
                                           IOCTL_PCIREGISTER,
                                           &pR,
                                           sizeof( sizeof( struct IoPciRegister ) ),
                                           &pR,
                                           sizeof( sizeof( struct IoPciRegister ) ),
                                           &BR,
                                           NULL);

        if( RetDevIo == FALSE )
                {
                DWORD dwErr;
                dwErr = GetLastError();
                return PCI_E_NAF_NOT_AVAILABLE;    // Error in GetProcAddress.
                }

        *PCIRegisterInfo = pR.PciRegisterInfo;
        *ExID = pR.ExID;

        iReturnCode = pR.iReturnCode;
```

```
        }

return iReturnCode;
}

//
// PciDeRegister()
//
PCI_INTEGER  PASCAL PciDeregister ( PCI_EXID  *ExID )
{
PCI_INTEGER      iReturnCode;

        CloseHandle( ExID->hDLLInstance );
        ExID->hDLLInstance = 0;

return iReturnCode;
}

//
// PciGetMessage()
//
PCI_INTEGER  PASCAL PciGetMessage ( PCI_EXID  *ExID,
                                    PCI_MPB  *PCIMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data )
{
PCI_INTEGER               iReturnCode;
DWORD                     BR;
BOOL                      RetDevIo;
struct   IoPciGetMessage  pGM;

if( ExID->hDLLInstance != INVALID_HANDLE_VALUE )
        {
        pGM.ExID = *ExID;
        pGM.PCIMPB = *PCIMPB;
        memcpy( pGM.Message , Message , COMMON_MAX_SIZE );
        memcpy( pGM.Data , Data , COMMON_MAX_SIZE );

        // Copy parameters to the input buffer.
        RetDevIo = DeviceIoControl(        ExID->hDLLInstance,
                                           IOCTL_PCIGETMESSAGE,
                                           &pGM,
                                           sizeof( struct IoPciGetMessage ),
                                           &pGM,
                                           sizeof( struct IoPciGetMessage ),
                                           &BR,
                                           NULL);

        if( RetDevIo == FALSE )
                {
                DWORD dwErr;
                dwErr = GetLastError();
                return PCI_E_NAF_NOT_AVAILABLE;     // Error in GetProcAddress.
                }

        *ExID = pGM.ExID;
        *PCIMPB = pGM.PCIMPB;
        memcpy( Message , pGM.Message , pGM.PCIMPB.MessageActualUsedSize );
        memcpy( Data , pGM.Data , pGM.PCIMPB.DataActualUsedSize );

        iReturnCode = pGM.iReturnCode;
        }

return iReturnCode;
}
```

H.6 NetWare implementation coding samples

H.6.1 Exchange mechanism functions

The following code shows a sample implementation of PUF exchange mechanism functions for the NetWare environment. The sample is illustrated using "C" language:

```
/*
 * Standard includes
 */
#include <process.h>
#include <advanced.h>
#include <string.h>

/*
 * Basic constants
 */
#define PCI_HANDLE_LENGTH              19
#define PCI_NAF_DRIVER_NAME_LENGTH     9
#define PCI_DATABASE_NAME              "SYS:\\PCI\\PCI.BTV"
#define PCI_DATABASE_PATH              "SYS:\\PCI\\"

#define PCI_GETPROPERTY_EXT_NAME       "_PciGetProperty"
#define PCI_REGISTER_EXT_NAME          "_PciRegister"
#define PCI_DEREGISTER_EXT_NAME        "_PciDeregister"
#define PCI_PUTMESSAGE_EXT_NAME        "_PciPutMessage"
#define PCI_GETMESSAGE_EXT_NAME        "_PciGetMessage"
#define PCI_GETSETSIGNAL_EXT_NAME      "_PciSetSignal"

#define PCI_E_SUCCESS                        0
#define PCI_E_INVALID_PARAMETER              99
#define PCI_E_INVALID_PCI_HANDLE             130
#define PCI_E_MAX_PUF_EXCEEDED               133
#define PCI_E_INVALID_PCI_EXID               136
#define PCI_E_DATA_BUFFER_TOO_SMALL          143
#define PCI_E_PROPERTY_BUFFER_TOO_SMALL      144
#define PCI_E_MORE_HANDLES                   147
#define PCI_E_NAF_NOT_AVAILABLE              255

/* some PCI's Messages definitions */
#define AManufacturerReq      111
#define AManufacturerInd      112

#define UDataReq              307
#define UDataInd              308


/*
 * Basic types
 */
typedef LONG PCI_INTEGER;
typedef BYTE * PCI_BYTEARRAY;
typedef BYTE PCI_HANDLE[PCI_HANDLE_LENGTH];
typedef LONG (*PCI_PROCEDURE)();
typedef struct {
   PCI_INTEGER   ExchangeId;
   PCI_PROCEDURE PciDeregisterPtr;
   PCI_PROCEDURE PciPutMessagePtr;
   PCI_PROCEDURE PciGetMessagePtr;
   PCI_PROCEDURE PciSetSignalPtr;
} PCI_EXID;

/*
 * PCI structures
 */
struct pci_mpb {
   PCI_INTEGER MessageID;
   PCI_INTEGER MessageMaximumSize;
   PCI_INTEGER MessageActualUsedSize;
   PCI_INTEGER DataMaximumSize;
   PCI_INTEGER DataActualUsedSize;
};
typedef struct pci_mpb PCI_MPB;

struct pci_register {
   PCI_INTEGER PUFVersion;
   PCI_INTEGER PUFType;
   PCI_INTEGER MaxMsgSize;
};

/*
 * PCI's functions prototypes
```

```
 */
extern PCI_INTEGER PciGetHandles(   PCI_INTEGER MaxHandles,
                                    PCI_BYTEARRAY PciHandles,
                                    PCI_INTEGER * ActualHandles);

extern PCI_INTEGER PciGetProperty(  PCI_HANDLE     PciHandle,
                                    PCI_INTEGER    MaximumSize,
                                    PCI_BYTEARRAY  Property,
                                    PCI_INTEGER    * ActualSize);

extern PCI_INTEGER PciRegister(     PCI_HANDLE           PciHandle,
                                    struct pci_register  * PciRegisterInfo,
                                    PCI_EXID             * ExId);

extern PCI_INTEGER PciDeregister(   PCI_EXID * ExId);

extern PCI_INTEGER PciPutMessage(   PCI_EXID * ExId,
                                    PCI_MPB * PciMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data);

extern PCI_INTEGER PciGetMessage(   PCI_EXID * ExId,
                                    PCI_MPB * PciMPB,
                                    PCI_BYTEARRAY Message,
                                    PCI_BYTEARRAY Data);

extern PCI_INTEGER PciSetSignal(    PCI_EXID * ExId,
                                    PCI_INTEGER Signal,
                                    PCI_PROCEDURE SignalProcedure);
/*
 * Btrieve declarations
 */
#define B_CREATE           14
#define B_OPEN             0
#define B_CLOSE            1
#define B_INSERT           2
#define B_DELETE           4
#define B_GET_EQUAL        5
#define B_GET_FIRST        12
#define B_GET_NEXT         6

#define DUP                1
#define MOD                2
#define BIN                4
#define SEG                16
#define EXT                256
#define NOCASE             1024

#define ZSTRING_KEY_TYPE   11

#define B_HANDLE_KEY_NUM   0

typedef struct {
   short int   KeyPos;
   short int   KeyLen;
   short int   KeyFlag;
   char        NotUse1[4];
   char        ExtKeyType;
   char        NullValue;
   char        Reserved1[2];
   char        ManualKeyNum;
   char        ACSNum;
} B_KEY_SPEC;

typedef struct {
   short int   RecLen;
   short int   PageSize;
   short int   NdxCnt;
   char        NotUse2[4];
   short int   FileFlag;
   char        Reserved2[2];
   short int   PreAlloc;
   B_KEY_SPEC  KeyBuf;
} B_FILE_SPEC;

typedef struct {
   char      NetAddress[12];
   char      NLMId[2];
   char      NLMClient[2];
} B_USER;

typedef struct {
   PCI_HANDLE     PciHandle;
   BYTE           PciNafDriverName[PCI_NAF_DRIVER_NAME_LENGTH];
} B_HANDLE_RECORD;
```

```
#define B_HANDLE_RECORD_LEN sizeof(B_HANDLE_RECORD)

extern WORD btrvID ( WORD        Operation,
                     void        * PositionBlock,
                     void        * DataBuf,
                     WORD        * DataLen,
                     void        * KeyBuf,
                     short int   KeyNum,
                     B_USER      * ClientID);


/*
 * ---------------------------- PciGetHandles -----------------------------------
 */
PCI_INTEGER PciGetHandles(PCI_INTEGER MaxHandles,
                          PCI_BYTEARRAY PciHandles,
                          PCI_INTEGER * ActualHandles)
{
   PCI_BYTEARRAY     Buffer;
   B_USER            User = {{0,0,0,0,0,0,0,0,0,0,0,0 }, 'P', 'C', 0, 0};
   B_HANDLE_RECORD   ClientBuf;
   WORD              Status;
   WORD              BufLen;
   PCI_HANDLE        RecordKey;
   char              ClientBlock[128];
   PCI_INTEGER       Ret;

   /* Check MaxHandles */
   if (MaxHandles < 1) {
      return(PCI_E_INVALID_PARAMETER);
   }

   /* Open the database */
   BufLen = 0;
   Status = btrvID(B_OPEN,
                   ClientBlock,
                   "",
                   &BufLen,
                   PCI_DATABASE_NAME,
                   0,
                   &User);

   /* if the database can't be opened, there is no NAF */
   *ActualHandles = 0;
   if (Status != 0) {
      return(PCI_E_SUCCESS);
   }

   /* Read the first record of the database */
   BufLen = B_HANDLE_RECORD_LEN;
   Buffer = PciHandles;
   Status = btrvID(B_GET_FIRST,
                   ClientBlock,
                   &ClientBuf,
                   &BufLen,
                   RecordKey,
                   B_HANDLE_KEY_NUM,
                   &User);

   /* Read all records if possible and store the handle */
   while((Status == 0) && (*ActualHandles < MaxHandles)) {
      memcpy(Buffer, RecordKey, PCI_HANDLE_LENGTH);
      Buffer += PCI_HANDLE_LENGTH;
      (*ActualHandles)++;

      /* Read the next record */
      BufLen = B_HANDLE_RECORD_LEN;
      Status = btrvID(B_GET_NEXT,
                      ClientBlock,
                      &ClientBuf,
                      &BufLen,
                      RecordKey,
                      B_HANDLE_KEY_NUM,
                      &User);
   }

   /* Compute the return code */
   if (Status == 0) {
      Ret = PCI_E_MORE_HANDLES;
   } else {
      Ret = PCI_E_SUCCESS;
   }

   /* Close the database */
```

```
   btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);

   /* End of function */
   return(Ret);
}

/*
 * ---------------------------- PciGetProperty ---------------------------------
 */
PCI_INTEGER PciGetProperty(PCI_HANDLE     PciHandle,
                           PCI_INTEGER    MaximumSize,
                           PCI_BYTEARRAY  Property,
                           PCI_INTEGER    * ActualSize)
{
   B_USER            User = {{0,0,0,0,0,0,0,0,0,0,0,0 }, 'P', 'C', 0, 0};
   B_HANDLE_RECORD   ClientBuf;
   WORD              Status;
   WORD              BufLen;
   char              ClientBlock[128];
   PCI_INTEGER       Ret;
   PCI_PROCEDURE     GetPropertyPtr;
   char              GetPropertyName[sizeof(PCI_GETPROPERTY_EXT_NAME) +
                                     PCI_NAF_DRIVER_NAME_LENGTH];

   /* Open the database */
   BufLen = 0;
   Status = btrvID(B_OPEN,
                   ClientBlock,
                   "",
                   &BufLen,
                   PCI_DATABASE_NAME,
                   0,
                   &User);

   /* if the database can't be opened, there is no NAF */
   if (Status != 0) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Read the record of the requested NAF */
   BufLen = B_HANDLE_RECORD_LEN;
   Status = btrvID(B_GET_EQUAL,
                   ClientBlock,
                   &ClientBuf,
                   &BufLen,
                   PciHandle,
                   B_HANDLE_KEY_NUM,
                   &User);

   /* Close the database */
   btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);

   /* If the record is not found */
   if (Status != 0) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Build PciGetProperty function's name */
   strcpy(GetPropertyName, ClientBuf.PciNafDriverName);
   strcat(GetPropertyName, PCI_GETPROPERTY_EXT_NAME);

   /* Link to the function */
   GetPropertyPtr = ImportSymbol(GetNLMHandle(), GetPropertyName);

   /* If link not possible, the NAF is not there */
   if (GetPropertyPtr == NULL) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Call the NAF's PciGetProperty function */
   Ret = (*GetPropertyPtr)(PciHandle,
                           MaximumSize,
                           Property,
                           ActualSize);

   /* Unlink function */
   UnimportSymbol(GetNLMHandle(), GetPropertyName);

   /* End of function */
   return(Ret);
}

/*
 * ---------------------------- PciRegister ------------------------------------
 */
```

```
PCI_INTEGER PciRegister(PCI_HANDLE           PciHandle,
                        struct pci_register  * PciRegisterInfo,
                        PCI_EXID             * ExId)
{
   B_USER            User = {{0,0,0,0,0,0,0,0,0,0,0,0 }, 'P', 'C', 0, 0};
   B_HANDLE_RECORD   ClientBuf;
   WORD              Status;
   WORD              BufLen;
   char              ClientBlock[128];
   PCI_INTEGER       Ret;
   PCI_PROCEDURE     RegisterPtr;
   char              RegisterName[sizeof(PCI_REGISTER_EXT_NAME) +
                                  PCI_NAF_DRIVER_NAME_LENGTH];

   /* Open the database */
   BufLen = 0;
   Status = btrvID(B_OPEN,
                   &ClientBlock[0],
                   "",
                   &BufLen,
                   PCI_DATABASE_NAME,
                   0,
                   &User);

   /* if the database can't be opened, there is no NAF */
   if (Status != 0) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Read the record of the requested NAF */
   BufLen = B_HANDLE_RECORD_LEN;
   Status = btrvID(B_GET_EQUAL,
                   ClientBlock,
                   &ClientBuf,
                   &BufLen,
                   PciHandle,
                   B_HANDLE_KEY_NUM,
                   &User);

   /* Close the database */
   btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);

   /* If the record is not found */
   if (Status != 0) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Build PciRegister function's name */
   strcpy(RegisterName, ClientBuf.PciNafDriverName);
   strcat(RegisterName, PCI_REGISTER_EXT_NAME);

   /* Link to the function */
   RegisterPtr = ImportSymbol(GetNLMHandle(), RegisterName);


   /* If link not possible, the NAF is not there */
   if (RegisterPtr == NULL) {
      return(PCI_E_NAF_NOT_AVAILABLE);
   }

   /* Call the NAF's PciRegister function */
   Ret = (*RegisterPtr)(PciHandle,
                        PciRegisterInfo,
                        ExId);

   /* Unlink function */
   UnimportSymbol(GetNLMHandle(), RegisterName);

   /* End of function */
   return(Ret);
}

/*
 * ---------------------------- PciDeregister -----------------------------------
 */
PCI_INTEGER PciDeregister(PCI_EXID * ExId)
{
   PCI_INTEGER Ret;

   /* Check if the PUF is registered */
   if (ExId->PciDeregisterPtr == 0) {
      return(PCI_E_INVALID_PCI_EXID);
   }

   /* Call the NAF's PciDeregister function */
```

```
   Ret = (*ExId->PciDeregisterPtr)(ExId);

   /* If deregister is successfull, unlink the functions' pointers */
   if (Ret == PCI_E_SUCCESS) {
      ExId->PciDeregisterPtr = NULL;
      ExId->PciPutMessagePtr = NULL;
      ExId->PciGetMessagePtr = NULL;
      ExId->PciSetSignalPtr = NULL;
   }

   /* Function's end */
   return(Ret);
}

/*
 * ---------------------------- PciPutMessage -----------------------------------
 */
PCI_INTEGER PciPutMessage(PCI_EXID * ExId,
                          PCI_MPB * PciMPB,
                          PCI_BYTEARRAY Message,
                          PCI_BYTEARRAY Data)
{
   PCI_INTEGER Ret;

   /* Check if the PUF is registered */
   if (ExId->PciPutMessagePtr == 0) {
      return(PCI_E_INVALID_PCI_EXID);
   }
   /* Call the NAF's PciPutMessage function */
   Ret = (*ExId->PciPutMessagePtr)(ExId, PciMPB, Message, Data);
   /* Function's end */
   return(Ret);
}

/*
 * ---------------------------- PciGetMessage -----------------------------------
 */
PCI_INTEGER PciGetMessage(PCI_EXID * ExId,
                          PCI_MPB * PciMPB,
                          PCI_BYTEARRAY Message,
                          PCI_BYTEARRAY Data)
{
   PCI_INTEGER Ret;

   /* Check if the PUF is registered */
   if (ExId->PciGetMessagePtr == 0) {
      return(PCI_E_INVALID_PCI_EXID);
   }
   /* Call the NAF's PciGetMessage function */
   Ret = (*ExId->PciGetMessagePtr)(ExId, PciMPB, Message, Data);
   /* Function's end */
   return(Ret);
}

/*
 * ---------------------------- PciSetSignal -----------------------------------
 */
PCI_INTEGER PciSetSignal(PCI_EXID * ExId,
                          PCI_INTEGER Signal,
                          PCI_PROCEDURE SignalProcedure)
{
   PCI_INTEGER Ret;

   /* Check if the PUF is registered */
   if (ExId->PciSetSignalPtr == NULL) {
      return(PCI_E_INVALID_PCI_EXID);
   }
   /* Call the NAF's PciSetSignal function */
   Ret = (*ExId->PciSetSignalPtr)(ExId, Signal, SignalProcedure);
   /* Function's end */
   return(Ret);
}
```

H.6.2 NAF declaration and extraction functions

The following code shows a sample implementation of NAF's declaration and extraction functions for the NetWare environment. The sample is illustrated using "C" language:

```
/*
 * This function suppose that a multiple load NAF has ALREADY check that the
 * handle is not used in a previous load of this driver
 */
PCI_INTEGER NAFDeclare(PCI_HANDLE PciHandle, char * DriverName)
{
   B_USER            User = {{0,0,0,0,0,0,0,0,0,0,0,0 }, 'P', 'C', 0, 0};
   B_FILE_SPEC       FileBuf;
   B_HANDLE_RECORD   ClientBuf;
   WORD              Status;
   WORD              BufLen;
   PCI_HANDLE        RecordKey;
   char              ClientBlock[128];
   PCI_INTEGER       Ret;
   DIR               * Dir;
   PCI_INTEGER       GetPropertyStatus;
   BYTE              NafProperty[128];
   PCI_INTEGER       ActualPropertySize;

   /* Try to open the Database */
   BufLen = 0;
   Status = btrvID(B_OPEN,
                   ClientBlock,
                   "",
                   &BufLen,
                   PCI_DATABASE_NAME,
                   0,
                   &User);

   /* if open fails */
   if (Status != 0) {

      /* Check if directory PCI exist */
      if ((Dir = opendir(PCI_DATABASE_PATH)) == 0) {
         /* if not, create it */
         if (mkdir(PCI_DATABASE_PATH) != 0) {
            return(-1);
         }
      } else {
         closedir(Dir);
      }

      /* Create the database */
      FileBuf.RecLen       = B_HANDLE_RECORD_LEN;
      FileBuf.PageSize     = 512;
      FileBuf.FileFlag     = 0;
      FileBuf.NdxCnt       = 1;
      FileBuf.Reserved2[0] = 0;
      FileBuf.Reserved2[1] = 0;

      FileBuf.KeyBuf.KeyPos         = 1;
      FileBuf.KeyBuf.KeyLen         = PCI_HANDLE_LENGTH;
      FileBuf.KeyBuf.KeyFlag        = EXT + NOCASE;
      FileBuf.KeyBuf.ExtKeyType     = ZSTRING_KEY_TYPE;
      FileBuf.KeyBuf.NullValue      = 0;
      FileBuf.KeyBuf.Reserved1[0]   = 0;
      FileBuf.KeyBuf.Reserved1[1]   = 0;
      FileBuf.KeyBuf.ManualKeyNum   = 0;
      FileBuf.KeyBuf.ACSNum         = 0;

      BufLen = sizeof(FileBuf);
      Status = btrvID ( B_CREATE,
                        ClientBlock,
                        &FileBuf,
                        &BufLen,
                        PCI_DATABASE_NAME,
                        0,
                        &User );

      /* if creation fails, return error */
      if (Status != 0) {
         return(-1);
      }

      /* Now, try to open the Database again */
      BufLen = 0;
      Status = btrvID(B_OPEN,
                      ClientBlock,
```

```
                  "",
                  &BufLen,
                  PCI_DATABASE_NAME,
                  0,
                  &User);

   /* if open fails, return error */
   if (Status != 0) {
      return(-1);
   }
}

/* Look for this handle in the database */
BufLen = B_HANDLE_RECORD_LEN;
Status = btrvID(B_GET_EQUAL,
                ClientBlock,
                &ClientBuf,
                &BufLen,
                PciHandle,
                B_HANDLE_KEY_NUM,
                &User);

/* if found, check the driver name */
if (Status == 0) {
   /* if the driver name is the same, the NAF is already declared */
   if (strncmp(ClientBuf.PciNafDriverName,
               DriverName,
               PCI_NAF_DRIVER_NAME_LENGTH) == 0) {
      /* Close the database */
      btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);

      /* return OK */
      return(PCI_E_SUCCESS);
   }

   /* The driver name is not the same, check if somebody else is declared */
   /* Call the GetProperty function of the other driver */
   GetPropertyStatus = PciGetProperty (PciHandle,
                                       sizeof(NafProperty),
                                       NafProperty,
                                       &ActualPropertySize);

   /* if successfull */
   if (GetPropertyStatus == PCI_E_SUCCESS) {
      /* Somebody else use this name, close the database */
      btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);

      /* return FAIL */
      return(-1);
   /* if not sucessfull */
   } else {
      /* Delete the record */
      BufLen = 0;
      Status = btrvID(B_DELETE,
                      ClientBlock,
                      &ClientBuf,
                      &BufLen,
                      RecordKey,
                      B_HANDLE_KEY_NUM,
                      &User);
   }
}

/* record the driver in the database */
BufLen = B_HANDLE_RECORD_LEN;
strncpy(ClientBuf.PciHandle, PciHandle, PCI_HANDLE_LENGTH);
strncpy(ClientBuf.PciNafDriverName, DriverName, PCI_NAF_DRIVER_NAME_LENGTH);
Status = btrvID(B_INSERT,
                ClientBlock,
                &ClientBuf,
                &BufLen,
                RecordKey,
                B_HANDLE_KEY_NUM,
                &User);

/* Check if write is successfull */
if (Status == 0) {
   Ret = PCI_E_SUCCESS;
} else {
   Ret = -1;
}

/* close the database */
btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);
```

```
   return(Ret);
}

PCI_INTEGER NAFExtract(PCI_HANDLE PciHandle)
{
   B_USER            User = {{0,0,0,0,0,0,0,0,0,0,0,0 }, 'P', 'C', 0, 0};
   B_HANDLE_RECORD   ClientBuf;
   WORD              Status;
   WORD              BufLen;
   PCI_HANDLE        RecordKey;
   char              ClientBlock[128];
   PCI_INTEGER       Ret;

   /* Open database */
   BufLen = 0;
   Status = btrvID(B_OPEN,
                   ClientBlock,
                   "",
                   &BufLen,
                   PCI_DATABASE_NAME,
                   0,
                   &User);

   if (Status != 0) {
      return(-1);
   }

   /* Look for the record in the database */
   BufLen = B_HANDLE_RECORD_LEN;
   Status = btrvID(B_GET_EQUAL,
                   ClientBlock,
                   &ClientBuf,
                   &BufLen,
                   PciHandle,
                   B_HANDLE_KEY_NUM,
                   &User);

   /* if record found */
   if (Status == 0) {
      /* Delete it */
      BufLen = 0;
      Status = btrvID(B_DELETE,
                      ClientBlock,
                      &ClientBuf,
                      &BufLen,
                      RecordKey,
                      B_HANDLE_KEY_NUM,
                      &User);

      if (Status == 0) {
         Ret = PCI_E_SUCCESS;
      } else {
         Ret = -1;
      }
   } else {
      Ret = -1;
   }
   /* Close the database */
   btrvID(B_CLOSE, ClientBlock, 0, 0, 0, 0, &User);
   return(Ret);
}
```

H.7 Windows 95 Operating System implementation coding samples

H.7.1 16 bits PUF

The coding samples for 16 bits PUF are descibed in annex H.2.

H.7.2 32 bits PUF

The following code shows a sample DLL implementation of PUF exchange functions for the Windows 95 environment. The sample is illustrated using "C" language:

```
// Standard includes
#include <windows.h>
#include "puf.h"


//
// PciGetHandles()
//

PCI_INTEGER PASCAL PciGetHandles(   PCI_INTEGER   MaxHandles,
                                    PCI_HANDLE    PCIHandles,
                                    PCI_INTEGER  *ActualHandles )
{
   INT                    nafNumber;
   INT                    nafFound;
   char                   keyName[20];
   PCI_BYTEARRAY          buffer;
   INT                    size;

   buffer = PCIHandles;
   for ( nafNumber=1, nafFound=0; nafNumber <= MaxHandles; nafNumber++) {
       wsprintf( keyName, « pciDriver%d », nafNumber);
       size = GetPrivateProfileString( « DRIVERS32 »,     // Section name
                                       keyName,           // pciDriver + 1..n
                                       NULL,              // no default string needed
                                       buffer,            // Address where to put the result
                                       256,               // Maxi.size for the result
                                       « PCI.INI »);      // INI filename

       if ( size > 0 ) {
           nafFound++;     // One more NAF found
           buffer+=256;    // Next location for a PCI handle (256 octets fixed size )
       }
   }
   *ActualHandles = nafFound;
   return PCI_E_SUCCESS;
}
```

```
//
// PciGetProperty()
//

PCI_INTEGER  PASCAL PciGetProperty (        PCI_HANDLE      PCIHandle,
                                            PCI_INTEGER     MaximumSize,
                                            PCI_INTEGER    *ActualSize,
                                            PCI_BYTEARRAY   Property)
{
   PCI_INTEGER    iReturnCode;
   HINSTANCE      hDLLInstance;
   FARPROC        lpfnGetProperty;

   // Load the NAF's DLL and return error if failed
   hDLLInstance = LoadLibrary(PCIHandle);
   if( hDLLInstance == NULL) return PCI_E_INVALID_PCI_HANDLE;

   // Get the "PciGetProperty" entry point of the dll.
   lpfnGetProperty = GetProcAddress( hDLLInstance , "PciGetProperty" );
   if( lpfnGetProperty == NULL ) {
       FreeLibrary( hDLLInstance );
       return PCI_E_NAF_NOT_AVAILABLE;      // Error in GetProcAddress.
   }

   // Call the "PciGetProperty" entry point of the dll.
   iReturnCode = lpfnGetProperty(PCIHandle,MaximumSize,Property,ActualSize);

   // Freelibrary in any case.
   FreeLibrary( hDLLInstance );
   return iReturnCode;
}


//
// PciRegister()
//

PCI_INTEGER  PASCAL PciRegister (   PCI_HANDLE               PCIHandle,
                                    struct pci_register     *PCIRegisterInfo,
                                    struct pci_opsys        *PCIOpSysInfo,
                                    PCI_EXID                *ExID)
{
   PCI_INTEGER    iReturnCode;
   HINSTANCE      hDLLInstance;
   FARPROC        lpfnRegister;

   // Load the NAF's DLL and return error if failed
   hDLLInstance = LoadLibrary(PCIHandle);
   if( hDLLInstance == NULL) return PCI_E_INVALID_PCI_HANDLE;
   ExID->hDLLInstance = hDLLInstance;

   // Get the "PciRegister" entry point of the dll.
   lpfnRegister = GetProcAddress( hDLLInstance , "PciRegister" );
   if( lpfnRegister == NULL ) {
      FreeLibrary( hDLLInstance );
      return PCI_E_NAF_NOT_AVAILABLE;       // Error in GetProcAddress.
   }

   // Call the "PciRegister" entry point of the dll.
   iReturnCode = lpfnRegister(PCIRegisterInfo, ExID);
   if( iReturnCode != 0 ) FreeLibrary( hDLLInstance );
   return iReturnCode;
}


//
// PciDeRegister()
//

PCI_INTEGER  PASCAL PciDeregister ( PCI_EXID  *ExID )
{
   PCI_INTEGER    iReturnCode;
   FARPROC        lpfnDeRegister;

   // Get the "PciDeRegister" entry point of the dll, return error if failed
   lpfnDeRegister = GetProcAddress( ExID->hDLLInstance , "PciDeRegister" );
   if( lpfnDeRegister == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
   ExID->hDLLInstance = 0;

   // Call the "Pci" entry point of the dll.
   iReturnCode = lpfnDeRegister( ExID );

   // FreeLibrary in any case.
   FreeLibrary( ExID->hDLLInstance );
   return iReturnCode;
```

```
}


//
// PciPutMessage()
//

PCI_INTEGER  PASCAL PciPutMessage ( PCI_EXID          *ExID,
                                    PCI_MPB           *PCIMPB,
                                    PCI_BYTEARRAY      Message,
                                    PCI_BYTEARRAY      Data)
{
   FARPROC        lpfnPutMessage;

   // Get the "PciPutMessage" entry point of the dll.
   lpfnPutMessage = GetProcAddress( ExID->hDLLInstance , "PciPutMessage" );
   if( lpfnPutMessage == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
   return lpfnPutMessage(ExID,PCIMPB,Message,Data);
}


//
// PciGetMessage()
//
PCI_INTEGER  PASCAL PciGetMessage ( PCI_EXID          *ExID,
                                    PCI_MPB           *PCIMPB,
                                    PCI_BYTEARRAY      Message,
                                    PCI_BYTEARRAY      Data)
{
   FARPROC        lpfnGetMessage;

   // Get the "PciGetMessage" entry point of the dll.
   lpfnGetMessage = GetProcAddress( ExID->hDLLInstance , "PciGetMessage" );
   if( lpfnGetMessage == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
   return lpfnGetMessage(ExID,PCIMPB,Message,Data);
}


//
// PciSetSignal()
//

PCI_INTEGER  PASCAL PciSetSignal (  PCI_EXID          *ExID,
                                    PCI_INTEGER        Signal,
                                    PCI_PROCEDURE      SignalProcedure)
{
   FARPROC        lpfnSetSignal;

   // Get the "PciSetSignal" entry point of the dll.
   lpfnSetSignal = GetProcAddress( ExID->hDLLInstance , "PciSetSignal" );
   if( lpfnSetSignal == NULL ) return PCI_E_NAF_NOT_AVAILABLE;
   return lpfnSetSignal(ExID,Signal,SignalProcedure);
}
```

H.7.3 VxD PUF

The following code shows a sample VxD services implementation of PUF exchange functions for the Windows95 environnement. The sample is illustrated using " C " language and inline assembler.

```
#include <puf.h>

#define    WANTVXDWRAPS

#include <c:\ddk\inc32\basedef.h>
#include <c:\ddk\inc32\vmm.h>
#include <c:\ddk\inc32\vmmreg.h>
#include <c:\ddk\inc32\debug.h>
#include <c:\ddk\inc32\vxdwraps.h>
#include <c:\ddk\inc32\vwin32.h>
#include <winerror.h>


// Declaration of the VxD services provided by PCI VxDs

#define VPCID_DEVICE_ID 0x18AC     // device id (example): should be taken from Microsoft

Begin_Service_Table       ( VPCID, VxD )
         Declare_Service   ( VPCID_GetVersion, LOCAL )
         Declare_Service   ( VPCID_MessageOperations, LOCAL )
End_Service_Table ( VPCID, VXD )

PCI_INTEGER VxDCallMessageOperation(BYTE iFunction)
{
   _asm mov ah, 0x02
   _asm mov al, iFunction

   VxDCall(VPCID_MessageOperations);
}

// The IsInfosPCI function checks if the registry value is a PCI value
// without used the standard C library

BOOL IsInfosPCI(char * Value)
{
   WORD i;
   char KeyName[10]="pciDriver";

   for (i=0;i<9;i++)
      if ( KeyName[i]!=Value[i] ) return FALSE;
   return TRUE;
}


PCI_INTEGER  PciGetHandles(        PCI_INTEGER        MaxHandles,
                                   PCI_BYTEARRAY      PCIHandles,
                                   PCI_INTEGER       *ActualHandles)
{
   DWORD ReturnCode=ERROR_SUCCESS, i=0;
   VMMHKEY hkResult;
   WORD NafFound=0,NafNumber;
   char ClassName[200];
   char ValueName[200];
   DWORD szClassName=200, MaxszClass;
   DWORD nbSubKeys, szSubKey;
   DWORD nbValues, szValueName=200;
   WORD szValueData, szData=sizeof(WORD);
   PCI_BYTEARRAY Data;
```

```
   Data = PCIHandles;
   Data[0]=0;

   if (_RegOpenKey(        HKEY_LOCAL_MACHINE,
                           "System\\CurrentControlSet\\Control\\Session Manager\\KnownVxDs",
                           &hkResult ) != ERROR_SUCCESS )
       return PCI_E_NAF_NOT_AVAILABLE;

   if ( _RegQueryInfoKey(  hkResult,
                           ClassName,
                           &szClassName,
                           NULL,
                           &nbSubKeys,
                           &szSubKey,
                           &MaxszClass,
                           &nbValues,
                           &szValueName,
                           &szValueData,
                           NULL,
                           NULL ) != ERROR_SUCCESS  )
       return PCI_E_NAF_NOT_AVAILABLE;

   ValueName[0]=0;
   for ( NafNumber = 1; ( (NafNumber <= MaxHandles) && (ReturnCode !=  ERROR_NO_MORE_ITEMS));
          NafNumber++ ) {
       ReturnCode = _RegEnumValue(  hkResult,
                                    i,
                                    (char *)ValueName,
                                    &szValueName,
                                    NULL,
                                    NULL,
                                    (unsigned char *)Data,
                                    &szData );
       if ( IsInfosPCI(ValueName) ) {
           NafFound++;
           Data+=128;
       }
       i++;
       szValueName=200;
       ValueName[0]=0;
   }
   _RegCloseKey(hkResult);
   *ActualHandles = NafFound;
   return PCI_E_SUCCESS;
}


PCI_INTEGER  PciGetProperty (       PCI_HANDLE         PCIHandle,
                                    PCI_INTEGER        MaximumSize,
                                    PCI_BYTEARRAY      Property,
                                    PCI_INTEGER       *ActualSize)
{
   PCI_INTEGER    ReturnCode;

   // Call GetVersion to verify the NAF VxD has been loaded
   VxDCall(VPCID_GetVersion);
   _asm jc END                      // If driver has not been loaded

   _asm mov ecx, PCIHandle
   _asm mov edx, dword ptr MaximumSize
   _asm mov esi,Property
   _asm mov edi,ActualSize
   VxDCallMessageOperation(PCI_SERVICE_GETPROPERTY);
   _asm mov ReturnCode,ax
   return ReturnCode;

END:
   return PCI_E_NAF_NOT_AVAILABLE;
}

PCI_INTEGER  PciRegister ( PCI_HANDLE                 PCIHandle,
                           struct pci_register       *PCIRegisterInfo,
                           struct pci_opsys          *PCIOpSysInfo,
                           PCI_EXID                  *ExID )
{
   PCI_INTEGER    ReturnCode;

   _asm mov ecx, PCIHandle
   _asm mov edx, PCIRegisterInfo
   _asm mov esi,  PCIOpSysInfo
   _asm mov edi,  ExID
   VxDCallMessageOperation(PCI_SERVICE_REGISTER);
   _asm mov ReturnCode,ax
   return ReturnCode;
```

```
}

PCI_INTEGER  PciDeregister (    PCI_EXID  *ExID )
{
   PCI_INTEGER    ReturnCode;

   _asm mov edi, ExID
   VxDCallMessageOperation(PCI_SERVICE_DEREGISTER);
   _asm mov ReturnCode,ax
   return ReturnCode;
}

PCI_INTEGER  PciPutMessage (        PCI_EXID          *ExID,
                                    PCI_MPB           *PCIMPB,
                                    PCI_BYTEARRAY      Message,
                                    PCI_BYTEARRAY      Data            )
{
   PCI_INTEGER    ReturnCode;

   _asm mov ecx, PCIMPB
   _asm mov edx, Message
   _asm mov esi, Data
   _asm mov edi, ExID
   VxDCallMessageOperation(PCI_SERVICE_PUTMESSAGE);
   _asm mov ReturnCode,ax
   return ReturnCode;
}

PCI_INTEGER PciGetMessage (         PCI_EXID          *ExID,
                                    PCI_MPB           *PCIMPB,
                                    PCI_BYTEARRAY      Message,
                                    PCI_BYTEARRAY      Data)
{
   PCI_INTEGER    ReturnCode;

   _asm mov ecx, PCIMPB
   _asm mov edx, Message
   _asm mov esi, Data
   _asm mov edi, ExID
   VxDCallMessageOperation(PCI_SERVICE_GETMESSAGE);
   _asm mov ReturnCode,ax
   return ReturnCode;
}

PCI_INTEGER  PciSetSignal (         PCI_EXID          *ExID,
                                    PCI_INTEGER       Signal,
                                    PCI_PROCEDURE     SignalProcedure)
{
   PCI_INTEGER    ReturnCode;

   _asm xor ecx, ecx       // Always NULL for a VxD
   _asm mov edx, SignalProcedure
   _asm mov edi, ExID
   VxDCallMessageOperation(PCI_SERVICE_SETSIGNAL);
   _asm mov ReturnCode,ax
   return ReturnCode;
}
```

Annex J (informative): TLV Coder/decoder sample

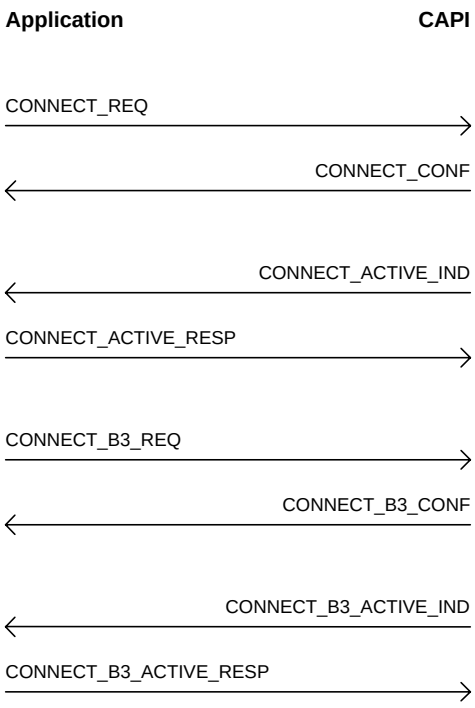
```
/*
//////////////////////////////////////////////////////////////////////
///
///     SAMPLES
///
///     TLV coder and decoder
///
//////////////////////////////////////////////////////////////////////
*/

#include <memory.h>
#include <stdarg.h>

/*
 * Definition of Types
 */
typedef int    BOOL;
#define FALSE      0
#define TRUE       1

#define LG_MAX_MESSAGE 128

/* Definition of structures */
struct sParameter /* Intermediate structure which receive the parameter to be added */
        {
        int iMessageLength;
        char scMessage[LG_MAX_MESSAGE];
        };

/*
//////////////////////////////////////////////////////////////////////
///
///     Function      : AddOctetParameter
///
///     Rule          : Add an octet parameter in a message
///
///     Parameters  :
///             structure sParameter pointer
///             parameter type
///             parameter value
///
///     Return        :
///             TRUE: Success
///             FALSE       : Error during processing
///
//////////////////////////////////////////////////////////////////////
*/
BOOL AddOctetParameter(     struct sParameter *pMessage, unsigned char cType, unsigned char
cValue)
        {
        if (pMessage->iMessageLength + 3 > LG_MAX_MESSAGE) /* Buffer is too small */
                {
                /* Process message size error */
                return FALSE;
                }/* if */
```

```
        /* TLV coding */
        pMessage->scMessage[pMessage->iMessageLength++] = cType;
        pMessage->scMessage[pMessage->iMessageLength++] = 1; /* length = 1 for octet */
        pMessage->scMessage[pMessage->iMessageLength++] = cValue; /* content */

        /* Success */
        return TRUE;
        }/* AddOctetParameter */

/*
//////////////////////////////////////////////////////////////////////
///
///     Function      : AddStringParameter
///
///     Rule   : Add a string (octet-string) parameter in a message
///
///     Parameters  :
///             structure sParameter pointer
///             parameter type
///             parameter length
///             parameter value (pointer)
///
///     Return        :
///             TRUE: Success
///             FALSE       : Error during processing
///
//////////////////////////////////////////////////////////////////////
*/
BOOL AddStringParameter(      struct sParameter *pMessage,
                              unsigned char cType,
                              int iLg,
                              unsigned char *lpValue)
        {
        if (iLg == 0) return FALSE;

        if (pMessage->iMessageLength + iLg + 2 > LG_MAX_MESSAGE) /* Buffer is too small */
                {
                /* Process message size error */
                return FALSE;
                }/* if */

        /* TLV coding */
        pMessage->scMessage[pMessage->iMessageLength++] = cType; /* Add the type */
        pMessage->scMessage[pMessage->iMessageLength++] = iLg; /* Length */
        memcpy(pMessage->scMessage+pMessage->iMessageLength, lpValue, iLg); /* Value */
        pMessage->iMessageLength += iLg;

        /* Success */
        return TRUE;
        }/* AddStringParameter */
```

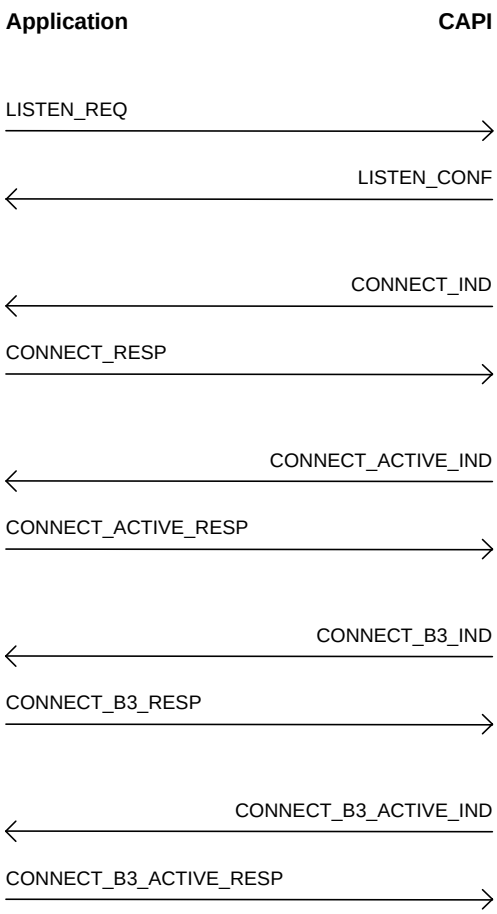
```
/*
////////////////////////////////////////////////////////////////////////
///
///     Function    : ExtractParameter
///
///     Rule        : Find a specific parameter and provide its location
///
///     Parameters  :
///             address to the message
///             current message length
///             parameter type we are looking for
///             pointer of pointer where to find value
///             pointer of an integer where to find the length of the parameter
///
///     Return      :
///             TRUE: Success
///             FALSE       : Error during processing
///
////////////////////////////////////////////////////////////////////////
*/
BOOL ExtractParameter( unsigned char *lpMessage,
                        unsigned int iLgMessage, unsigned char cType,
                        unsigned char * *lplpValue, unsigned int *lpiLgValue)
        {
        while (iLgMessage > 0) /* for all message parameters */
                {
                if (*lpMessage != cType)
                        {
                        /* process the next parameter */
                        iLgMessage -= lpMessage[1] + 2;
                        lpMessage += lpMessage[1] + 2;
                        continue;
                        }/* if */

                /* the parameter type is found update information for the caller */
                *lplpValue = lpMessage + 2;
                *lpiLgValue = lpMessage[1];

                /* Success */
                return TRUE;
                }/* while */

        return FALSE;
        }/* ExtractParameter */
```

Annex K (informative): Sample flow chart diagrams of Profile B

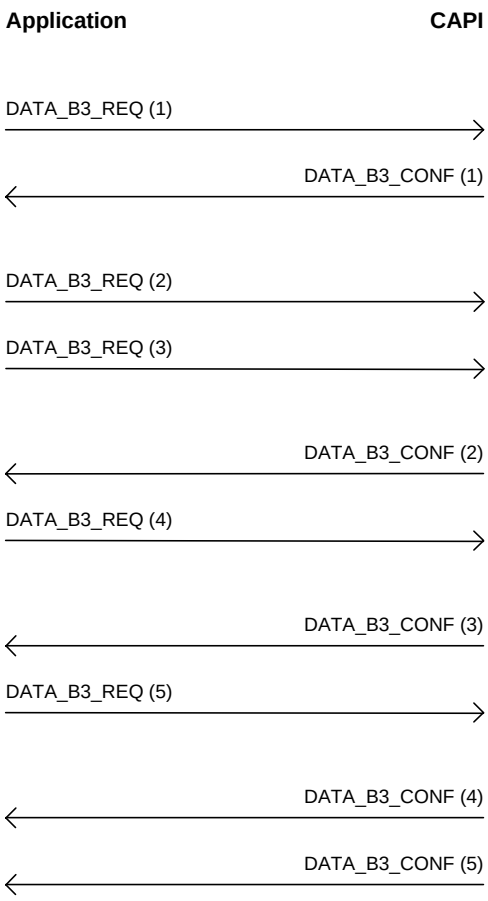
K.1 Outgoing call



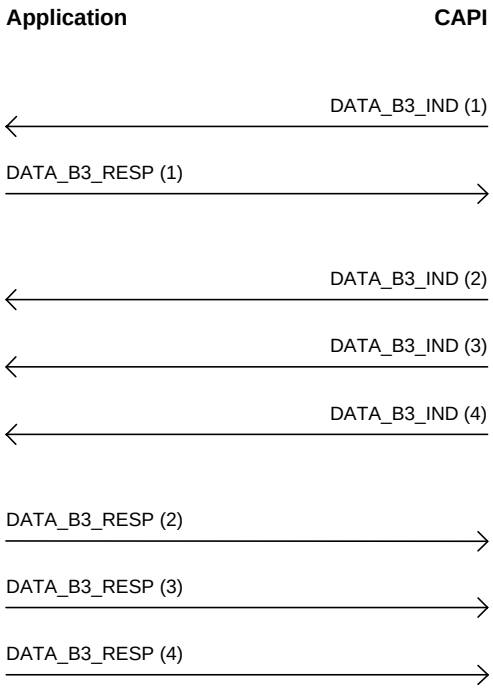
K.2 Incoming call



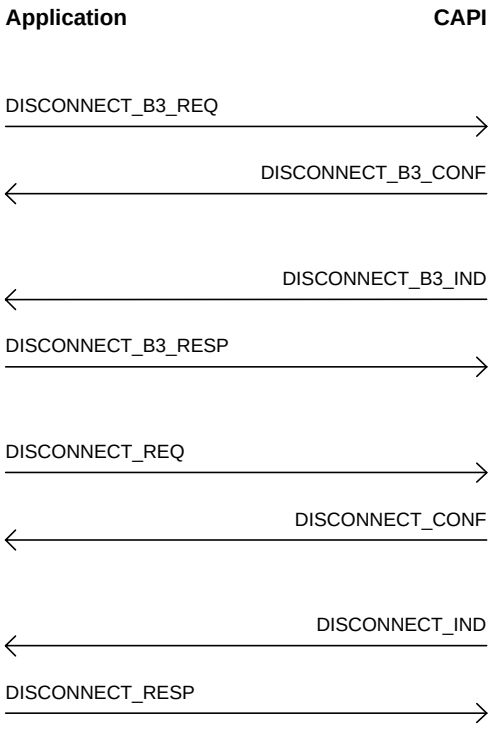
K.3 Transmitting data



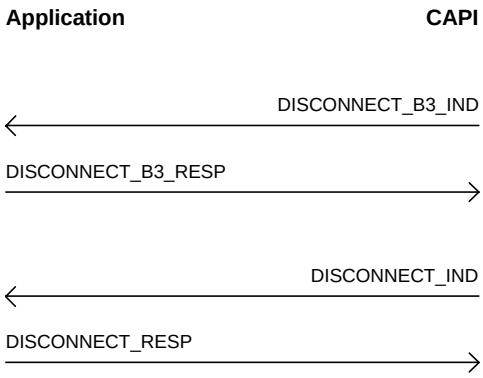
K.4 Receiving data



K.5 Active disconnect

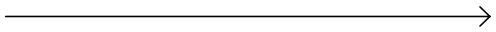


K.6 Passive disconnect

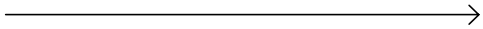


K.7 Disconnect collision

Simultaneous release of a physical connection by application and Profile B



also possible:



illegal:

after DISCONNECT_IND no more messages are sent to applications, so DISCONNECT_REQ is not be confirmed

after DISCONNECT_IND no more messages are sent to applications, so DISCONNECT_REQ is not be confirmed

invalid, after DISCONNECT_IND no more message concerning this PLCI are sent to application

K.8 X.25 D-channel

For X.25 in the D-channel the configuration and establishment of layer 2 is accomplished by a CONNECT_REQ message with the parameter

B-Protocol.B2-Protocol=LAPD
B-Protocol.B2-Configuration.AddressA=TEI
B-Protocol.B3-Protocol=ISO8208

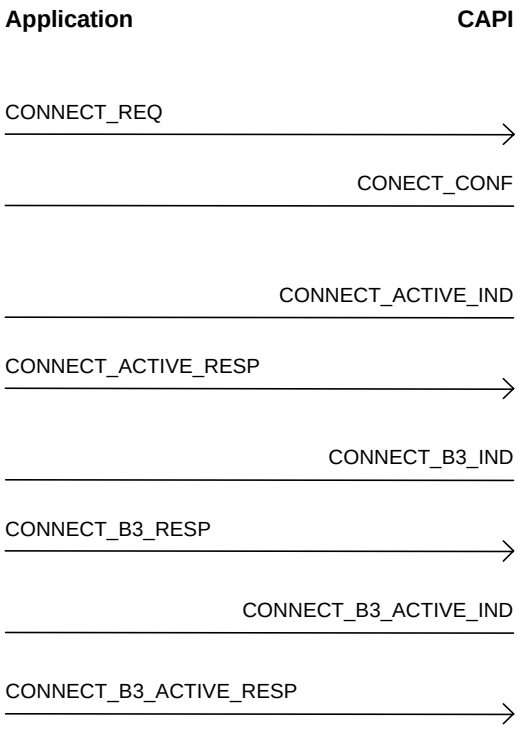
Parameters that are not relevant for X.25 in D-channel are ignored by Profile B. The address of the called party shall be passed in the X.25 call packet which is part of the NCPI parameter of the CONNECT_B3_REQ message.

The passive connection is initiated by a CONNECT_REQ message, no LISTEN_REQ and consequently no CONNECT_IND message is sent.

EXAMPLES:

Active connection setup: no difference compared to other protocols, see figure A.1.

Passive connection setup:



Annex L (normative): SFF Format (Profile B)

L.1 Introduction

SFF (**S**tructured **F**ax **F**ile) is a representation specially for facsimile group 3 documents. It consists of information concerning the page structure and compressed line data of the facsimile document. A SFF formatted document always starts with a header, valid for the complete document. Every page starts with a page header. After this the pixel information follows line by line. As the SFF format is a file format specification, some entries in header structures (e.g. double chaining of pages) may not used or supported by Profile B.

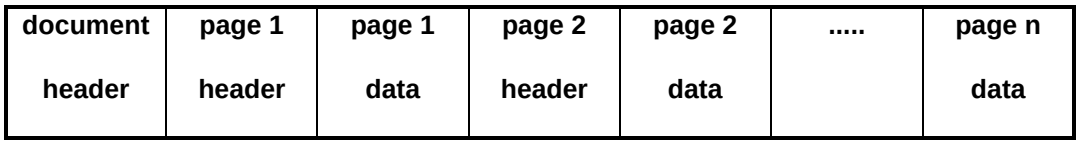


Figure L.1: SFF format

L.2 SFF coding rules

Following type conventions are used:

byte	8 bit unsigned
word	16 bit unsigned integer, least significant octet first
dword	32 bit unsigned integer, least significant word first

L.2.1 Document header

Parameter	Type	Comment
SFF_Id	dword	magic value (identification) of SFF Format: coded as 0x66666653 ("SFFF")
Version	byte	version number of SFF document: coded 0x01
reserved	byte	reserved for future extensions, coded 0x00
User Information	word	manufacturer specific user information (not used by Profile B, coded as 0x0000)
Page Count	word	number of document's pages. If not known (in case of receiving a document) it shall be coded 0x0000.
OffsetFirstPageHeader	word	byte offset of first page header from start of document header. This value is normally equal to the size of the document header (0x14), but there might be additional user specific data between document header and first page header. Profile B shall ignore and not offer such additional data.
OffsetLastPageHeader	dword	byte offset of last page header from start of document header. If not known (in case of receiving a document) it shall be coded 0x00000000.
OffsetDocumentEnd	dword	byte offset to document end from start of document header. If not known (in case of receiving a document) it shall be coded 0x00000000.

L.2.2 Page header

Parameter	Type	Comment
PageHeaderID	byte	254 (Record Type of Page Data)
PageHeaderLen	byte	0: Document end 1..255: byte offset of first page data from entry *Resolution Vertical* of page header. This value is normally equal to the size of the following part of the header (0x10), but there might be additional user specific data between page header and page data. Profile B shall ignore and not offer such additional data.
Resolution Vertical	byte	definition of vertical resolution; different resolutions in one document may be ignored by Profile B. 0: 98 lpi (standard) 1:: 196 lpi (high resolution) 2..254: reserved 255: end of document (should not be used, instead *PageHeaderLen* should be coded 0)
Resolution Horizontal	byte	definition of horizontal resolution 0: 203 dpi (standard) 1..255: reserved
Coding	byte	definition of pixel line coding 0: modified Huffman coding 1..255: reserved
reserved	byte	coded as 0
Line Length	word	number of pixels per line 1728: standard fax g3 2048: B4 (optional) 2432: A3 (optional) Support of other values also is optional for Profile B.
Page Length	word	number of pixel lines per page. If not known, coded as 0x0000.
OffsetPreviousPage	dword	byte offset to previous page header or 0x00000000. Coded as 0x00000001 if first page.
OffsetNextPage	dword	byte offset to next page header or 0x00000000. Coded as 0x00000001 if last page.

L.2.3 Page data

Page data is coded line by line, i.e. for each pixel row exists a data definition. Lines are coded as records with variable length, each line is coded according to element *coding* in page header. For the moment only modified Huffman coding is supported. MH-coding is byte oriented, the first bit or a code word is stored least significant first. There are no EOL code words or fill bits included. If data include EOL code words, Profile B shall ignore these coding.

Each record is identified by the first byte:

- **1..216:** pixel row with 1..216 MH-coded bytes are following immediately;

- **0:** escape for pixel row with more than 216 bytes MH-coding. In this case, a following word in the range **217..32 767** defines the number of MH-coded bytes, which are following;

- **217..253**: white skip, 1..37 empty lines;

- **254**: start or page header (see there);

- **255:** if followed by a byte with value **0**, illegal line coding. An application can decide if this line should be interpreted empty or as a copy of the previous line. If this byte is followed by a byte with a value **1..255**, 1..255 bytes additional user information are following (reserved for future extensions).

Annex M (informative): Protocols supported by Profile B

Profile B provides a standardized interface for any number of application programs (applications) to any number of ISDN drivers and ISDN controllers. Applications can be freely assigned to drivers and controllers:

- one application can use one controller
- one application can use more than one controller
- several applications can share a single controller
- several applications can share more than one controller

Applications can use different protocols at different protocol levels, Profile B provides a selection mechanism in support of this. Profile B also performs an abstraction from different protocol variants, creating a standardized network access. All connection related data such as connection state, display messages, etc., is available to applications at any time.

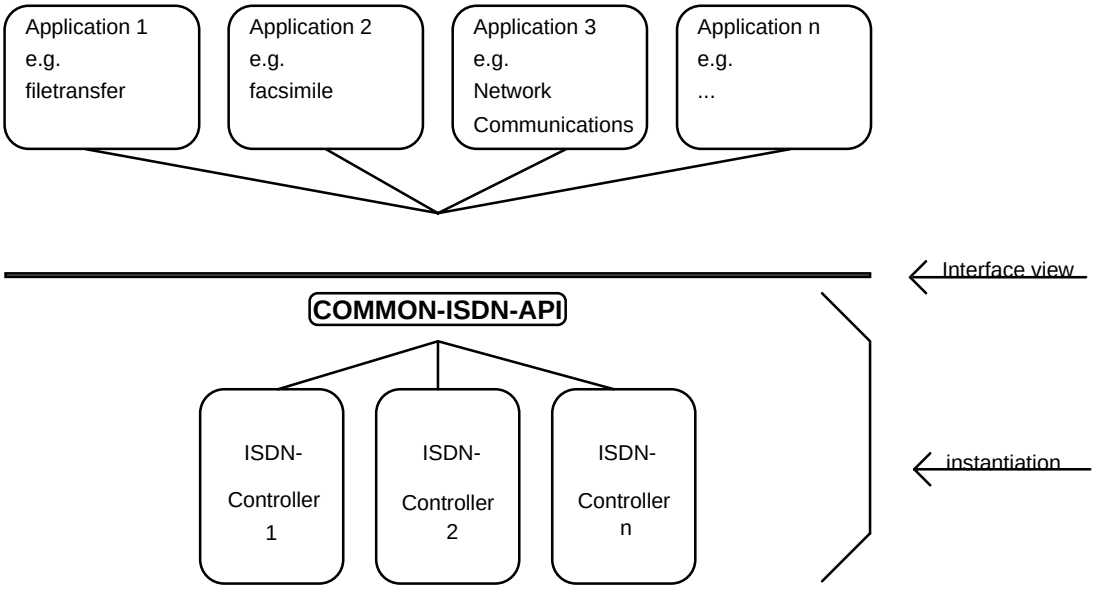


Figure M.1: Position of Profile B

Profile B covers the whole signalling protocol as well as protocol layers 1 to 3 (physical and framing layer, data link layer and network layer) for data channels. The interface of Profile B is located between layer 3 and layer 4 and provides the point of reference for applications and higher level protocols.

Profile B offers many currently used protocols to applications without deep protocol knowledge. The default protocol is **ISO 7776 [4]** (X.75 SLP), i.e. framing protocol **HDLC**, data link protocol **ISO 7776 [4]** (**X.75** SLP), and a transparent network layer.

Other supported variants of framing layer are: **HDLC inverted**, **PCM** (bit transparent with byte framing) **64/56** kBit, **V.110 [17]** sync / async. Profile B integrates the following data link and network layers: **LAPD** according to Q.921 [13] for **X.25 D-channel** implementation, **PPP [10] [11]** (Point to Point protocol), **ISO 8208 [3]** (**X.25** DTE-DTE), **X.25 DCE**, **T.90NL** (with compatibility to **T.70NL**) and **T.30 [14]** (facsimile group 3).

Even if not all protocols can fit completely within the OSI scheme, Profile B always supports three layers. Each layer can be configured by applications. In case of illegal or meaningless combinations of protocol stack combinations (e.g. bit transparency 56 kBits and X.25 DCE) Profile B shall report this error.

Annex N (informative): Development guidelines for Profile B

The main body of this ETS contains the description of Profile B from the application point of view. Following this approach, certain points, not directly related to the application, which have an impact on the development of the implementation of Profile B are not described. These points may be of interest for the implementation development and are, therefore, described in Profile B, subclause 6.9 in the state diagrams. This annex gives these guidelines for the development of the implementation in accordance with ITU-T Recommendation Z.100 [5].

N.1 SDL diagrams

N.1.1 SDL diagrams: conventions

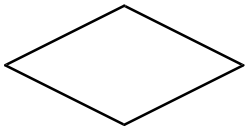
The SDL diagrams are given to explain more clearly the relation between Profile B messages and Profile B actions. Note that Profile B does not define a direct mapping between Profile B messages and network primitives. Instead Profile B offers functionalities, which are independent of the used network protocol.

The following symbols are used within this description. A full description of the symbols and their meaning is given in ITU-T Recommendation Z.100 [5].

State Symbol

Input (from NAF implementation)

Input (from PUF)



Decision Symbol

Output (to NAF implementation)

Output (to PUF)

N.1.2 SDL diagrams for Control Plane

The following SDL diagrams show the internal states of the signalling protocol section of Profile B. The primitives shown in upper case are messages defined by Profile B.

NOTE: Invalid input from an application shall not result in a state transition. In case of Requests from the PUF a Confirmation shall be send to the PUF to indicate the invalid request. Invalid Responses shall be ignored.

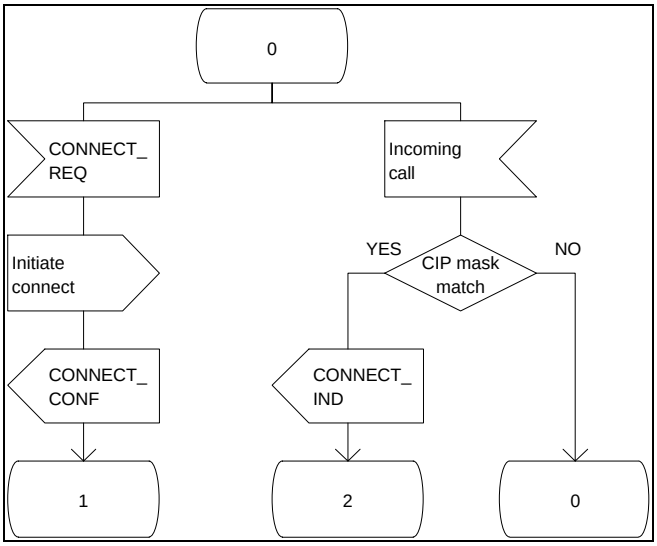


Figure N.1

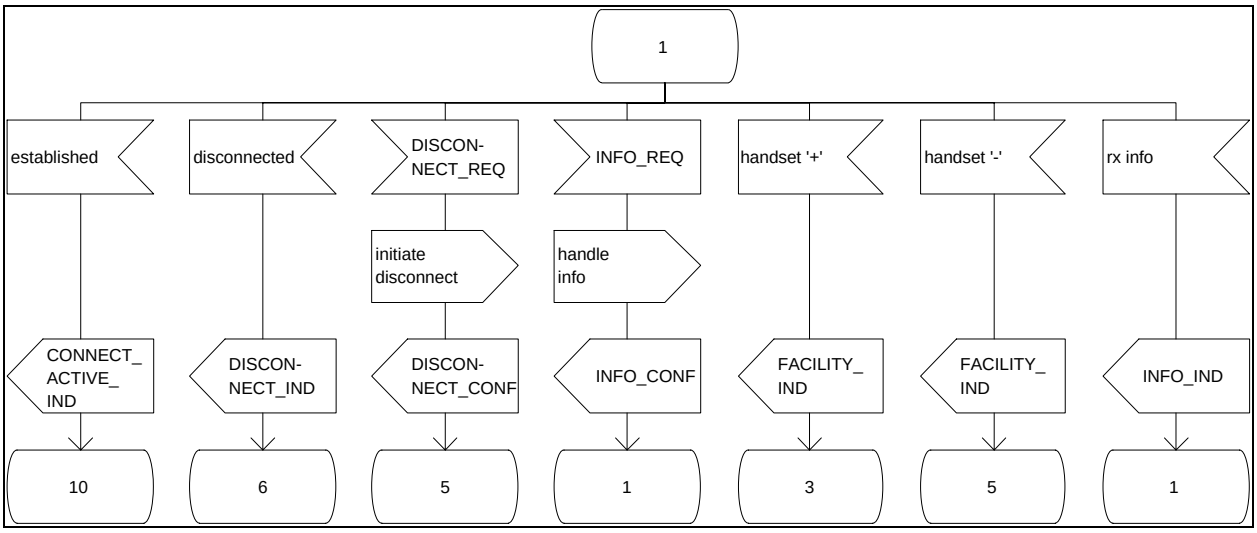


Figure N.2

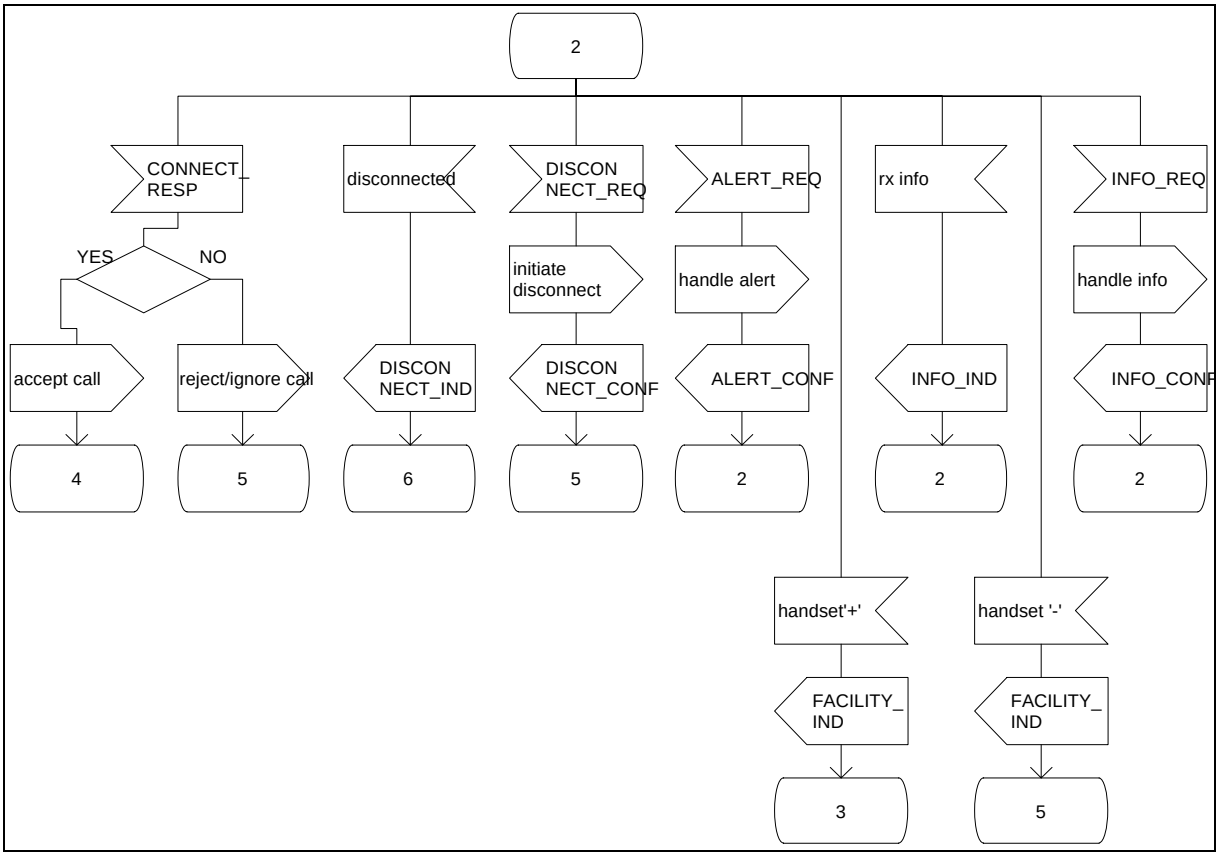


Figure N.3

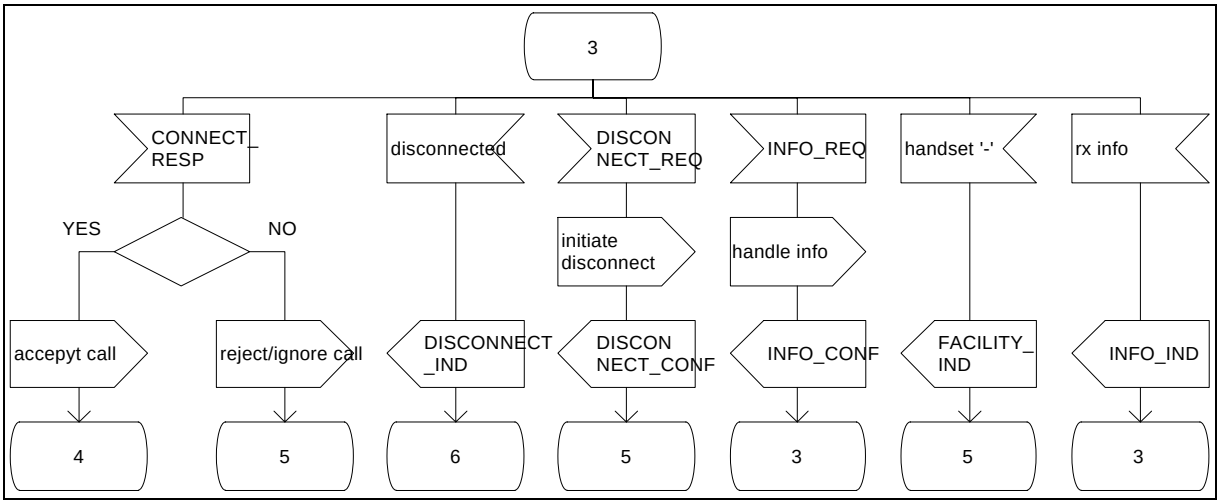


Figure N.4

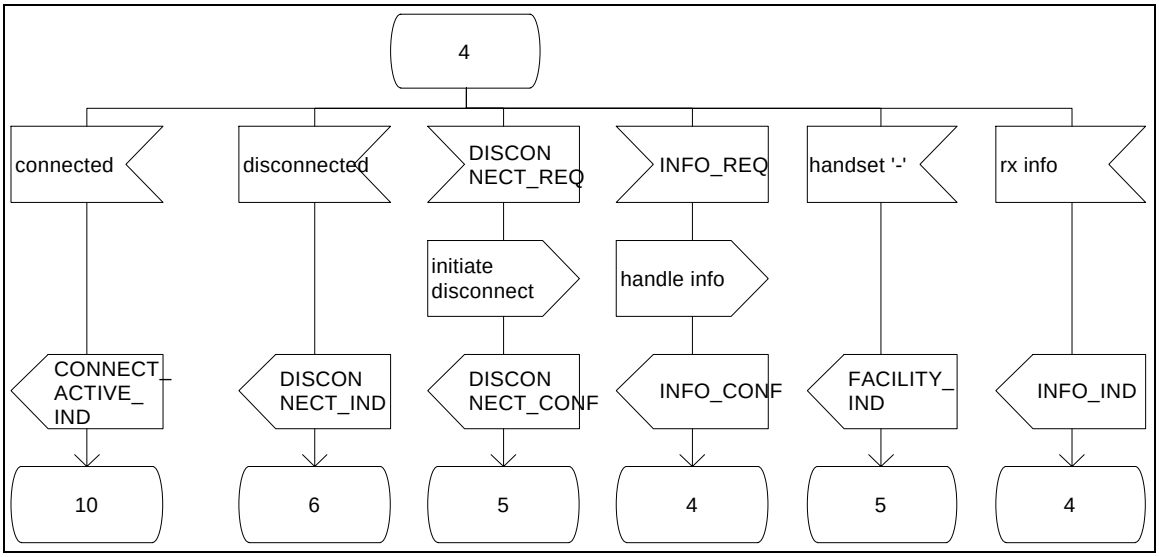


Figure N.5

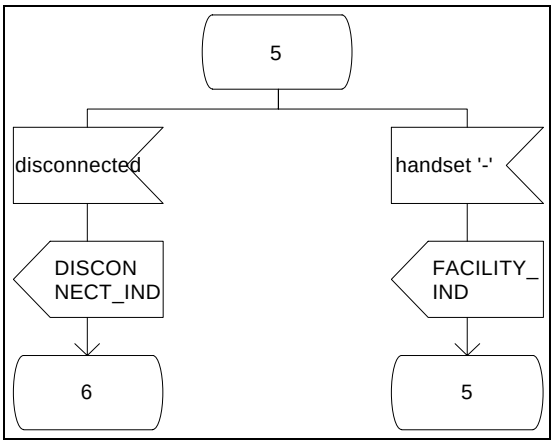


Figure N.6

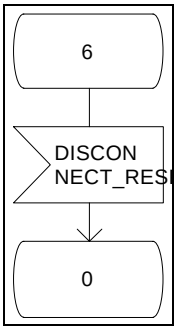


Figure N.7

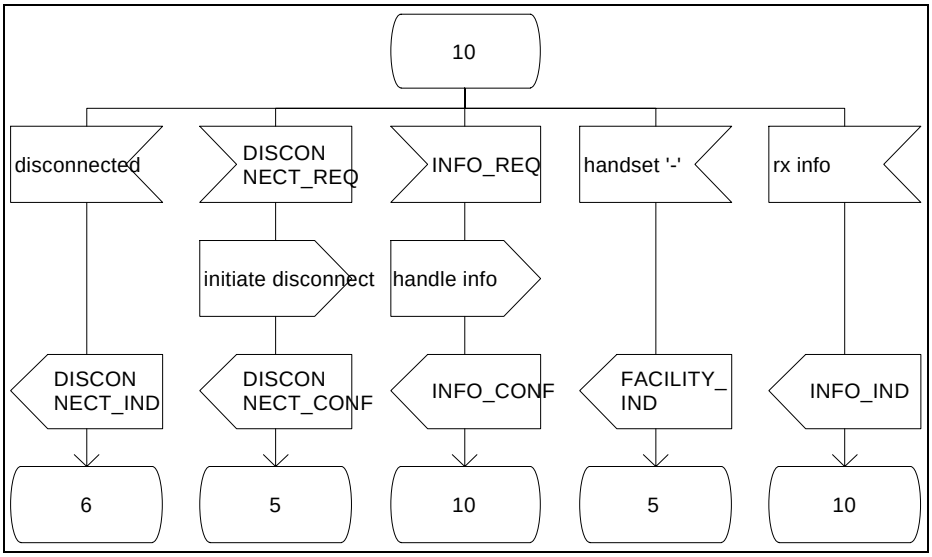


Figure N.8

N.1.3 SDL diagrams for User Plane

The following SDL diagrams show the internal states of the logical connection section of Profile B. The primitives shown in upper case are messages defined by Profile B.

NOTE: Invalid input from PUF shall not result in a state transition. In case of Requests from the PUF a Confirmation shall be send to the PUF to indicate the invalid request. Invalid Responses shall be ignored.

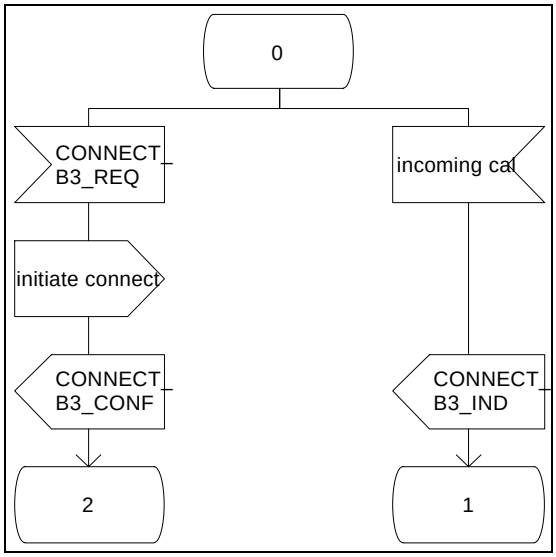


Figure N.9

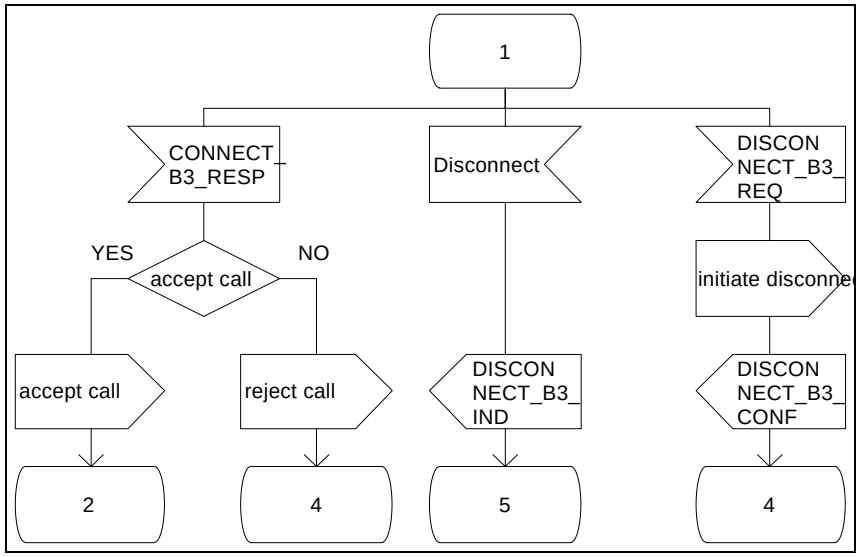


Figure N.10

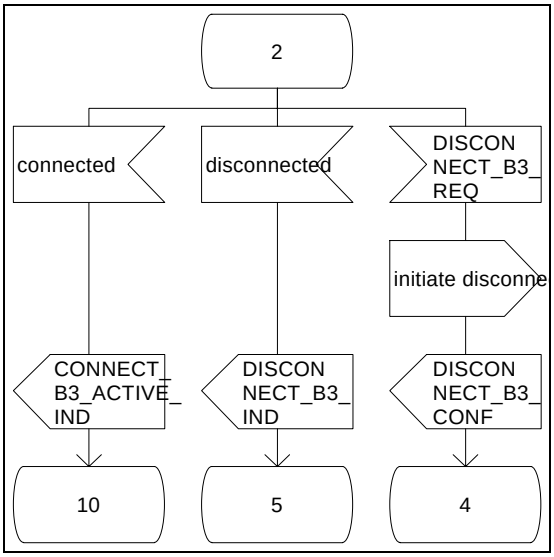


Figure N.11

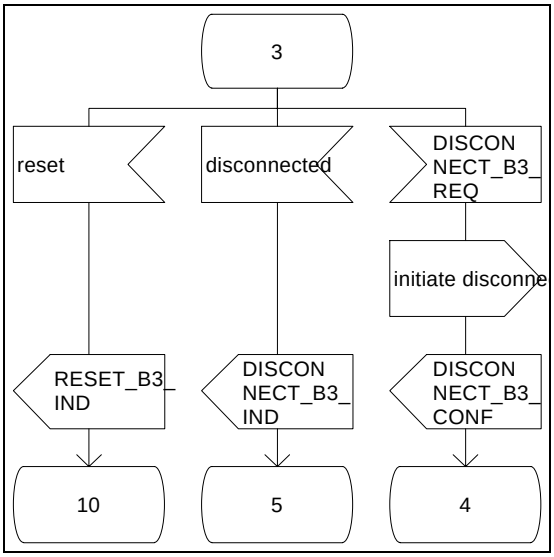


Figure N.12

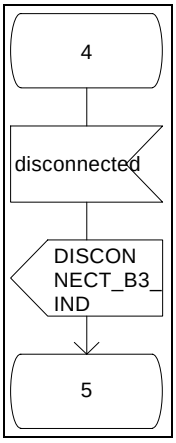


Figure N.13

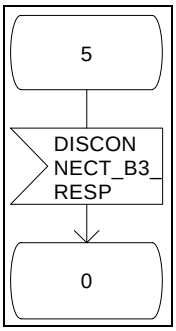


Figure N.14

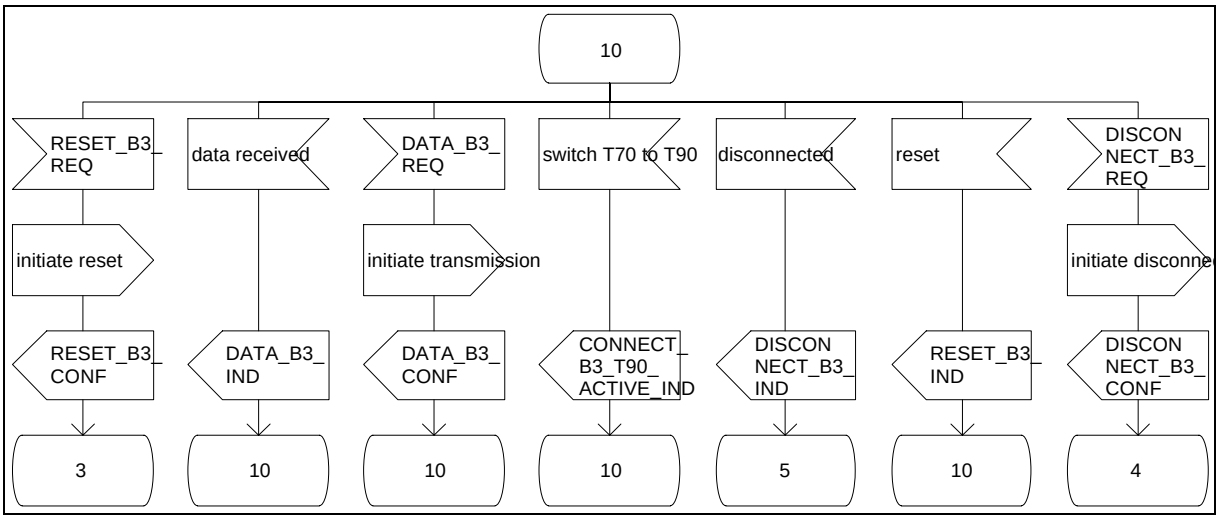


Figure N.15

Annex P (informative): Profile B Implementation description

P.1 Introduction

This annex contains the implementation description of Profile B. It lists all mandatory, conditional and optional items of the specification for Profile B relating to the exchange mechanism and the supported messages. It shall be used in the process of evaluating a particular implementation when claiming conformance to or support of the specification of Profile B. The implementation which claims conformance can either be an implementation of Profile B or an application

To evaluate conformance of a particular implementation, it is necessary to have a statement of which capabilities and options have been implemented. This annex contains such a statement.

P.2 How to read the following tables

Each line within the following tables is numbered at the left hand edge of the line. This numbering is included as a means of uniquely identifying all possible implementation details within Profile B.

The I/A column in this annex separates the capabilities for the implementation of Profile B and the application using this Profile.

I Implementation of Profile B

A Application using this profile

The I/A-column can be omitted from the Administration Plane, Control Plane and User Plane items as all those items are optional on the application level.

The D column in this annex reflects the definition of the items in this ETS. Each entry in this column is chosen from the following list:

M Mandatory support is required;

O Optional support is permitted. If implemented, it shall conform to this ETS;

The I column in this annex describes the actual capabilities of the implementation and shall be completed by the supplier using a symbol chosen from the following list:

Y item is implemented (subject to stated constraints);

N item is not implemented;

- item is not applicable.

The ref./note column in this annex contains the references to the location in the main body of this ETS where the items are described or a note explaining why the item is conditional.

The following abbreviations are used in the headings of this annex:

I/A Implementation / Application

D Defined.

I Implemented.

ref. reference.

P.3 Exchange mechanism

	Item of Profile B	I/A	D	I	ref./note
1	CAPI_REGISTER	I	M	[ ]	6.3.2.1
		A	M	[ ]	6.3.2.1
2	CAPI_RELEASE	I	M	[ ]	6.3.2.4
		A	M	[ ]	6.3.2.4
3	CAPI_PUT_MESSAGE	I	M	[ ]	6.3.2
		A	M	[ ]	6.3.2
4	CAPI_GET_MESSAGE	I	M	[ ]	6.3.2
		A	M	[ ]	6.3.2
5	CAPI_SET_SIGNAL	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2
6	CAPI_GET_MANUFACTURER	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2
7	CAPI_GET_VERSION	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2
8	CAPI_GET_SERIAL_NUMBER	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2
9	CAPI_GET_PROFILE	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2
10	CAPI_MANUFACTURER	I	M	[ ]	6.3.2
		A	O	[ ]	6.3.2

P.4 Administration Plane

	Item of Profile B	D	I	ref./note
11	LISTEN_REQ	M	[ ]	6.7.37
12	LISTEN_CONF	M	[ ]	6.7.38
13	FACILITY_REQ	M	[ ]	6.7.29
14	FACILITY_CONF	M	[ ]	6.7.30
15	FACILITY_IND	M	[ ]	6.7.31
16	FACILITY_RESP	M	[ ]	6.7.32
17	SELECT_B_PROTOCOL_REQ	M	[ ]	6.7.47
18	SELECT_B_PROTOCOL_CONF	M	[ ]	6.7.48

P.5 Control Plane

	Item of Profile B	D	I	ref./note
19	CONNECT_REQ	M	[ ]	6.7.3
20	CONNECT_CONF	M	[ ]	6.7.4
21	CONNECT_IND	M	[ ]	6.7.5
22	CONNECT_RESP	M	[ ]	6.7.6
23	CONNECT_ACTIVE_IND	M	[ ]	6.7.7
24	CONNECT_ACTIVE_RESP	M	[ ]	6.7.8
25	DISCONNECT_REQ	M	[ ]	6.7.25
26	DISCONNECT_CONF	M	[ ]	6.7.26
27	DISCONNECT_IND	M	[ ]	6.7.27
28	DISCONNECT_RESP	M	[ ]	6.7.28
29	ALERT_REQ	M	[ ]	6.7.1
30	ALERT_CONF	M	[ ]	6.7.2

	Item of Profile B	D	I	ref./note
31	INFO_REQ	M	[ ]	6.7.33
32	INFO_CONF	M	[ ]	6.7.34
33	INFO_IND	M	[ ]	6.7.35
34	INFO_RESP	M	[ ]	6.7.36

P.6 User Plane

	Item of Profile B	D	I	ref./note
35	CONNECT_B3_REQ	M	[ ]	6.7.11
36	CONNECT_B3_CONF	M	[ ]	6.7.12
37	CONNECT_B3_IND	M	[ ]	6.7.13
38	CONNECT_B3_RESP	M	[ ]	6.7.14
39	CONNECT_B3_ACTIVE_IND	M	[ ]	6.7.9
40	CONNECT_B3_ACTIVE_RESP	M	[ ]	6.7.10
41	CONNECT_B3_T90_ACTIVE_IND	M	[ ]	6.7.15
42	CONNECT_B3_T90_ACTIVE_ RESP	M	[ ]	6.7.16
43	DISCONNECT_B3_REQ	M	[ ]	6.7.21
44	DISCONNECT_B3_CONF	M	[ ]	6.7.22
45	DISCONNECT_B3_IND	M	[ ]	6.7.23
46	DISCONNECT_B3_RESP	M	[ ]	6.7.24

	Item of Profile B	**D**	**I**	**ref./note**
47	DATA_B3_REQ	M	[ ]	6.7.17
48	DATA_B3_CONF	M	[ ]	6.7.18
49	DATA_B3_IND	M	[ ]	6.7.19
50	DATA_B3_RESP	M	[ ]	6.7.20
51	RESET_B3_REQ	M	[ ]	6.7.43
52	RESET_B3_CONF	M	[ ]	6.7.44
53	RESET_B3_IND	M	[ ]	6.7.45
54	RESET_B3_RESP	M	[ ]	6.7.46
55	MANUFACTURER _REQ	M	[ ]	6.7.39
56	MANUFACTURER _CONF	M	[ ]	6.7.40
57	MANUFACTURER _IND	M	[ ]	6.7.41
58	MANUFACTURER _RESP	M	[ ]	6.7.42

P.7 User Plane protocols

P.7.1 User Plane B1 protocols

The *B1 protocols* specify the physical layer and framing used for the connection.

	Item of Profile B	**D**	**I**	**ref./note**
59	64 kBit/s with HDLC framing Default B1 protocol	M	[ ]	6.8
60	64 kBit/s bit transparent operation with byte framing from the network	O	[ ]	6.8
61	V.110 [17] asynchronous operation with start/stop byte framing	O	[ ]	6.8
62	V.110 [17] synchronous operation with HDLC framing	O	[ ]	6.8
63	T.30 [14] modem for fax group 3	O	[ ]	6.8
64	64 kBit/s inverted with HDLC framing	O	[ ]	6.8
65	56 kBit/s bit transparent operation with byte framing from the network	O	[ ]	6.8

P.7.2 User Plane B2 protocols

The *B2 protocols* specify the data link layer used for the connection.

	Item of Profile B	**D**	**I**	**ref./note**
66	ISO 7776 [4] (X.75 SLP) This is the default B2 protocol	M	[ ]	6.8
67	Transparent	O	[ ]	6.8
68	SDLC [12]	O	[ ]	6.8
69	LAPD according Q.921 [13] for D channel X.25	O	[ ]	6.8
70	T.30 [14] for fax group 3	O	[ ]	6.8
71	Point to Point Protocol (PPP [10] [11])	O	[ ]	6.8
72	Transparent (ignoring framing errors of B1 protocol)	O	[ ]	6.8

P.7.3 User Plane B3 protocols

The *B3 protocols* specify the network layer used for the connection.

	Item of Profile B	**D**	**I**	**ref./note**
73	Transparent. This is the default B3 protocol	M	[ ]	6.8
74	T.90NL with compatibility to T.70NL according to T.90 Appendix II [15] [16].	O	[ ]	6.8
75	ISO 8208 [3] (X.25 DTE-DTE)	O	[ ]	6.8
76	X.25 DCE.	O	[ ]	6.8
77	T.30 [14] for fax group 3	O	[ ]	6.8

Annex Q (informative): Index of Profile B related topics

History

Document history			
March 1993	Public Enquiry	PE 40:	1993-03-15 to 1993-08-06
March 1994	First Edition		
January 1996	Public Enquiry	PE 100:	1996-01-22 to 1996-05-17
May 1996	Converted into Adobe Acrobat Portable Document Format (PDF)		