EUROPEAN TELECOMMUNICATION STANDARD

FINAL DRAFT
pr **ETS 300 287-1**

September 1996

Second Edition

Source: ETSI TC-SPS

Reference: RE/SPS-02035

ICS: 33.080

Key words: ISDN, SS7, TCAP

Integrated Services Digital Network (ISDN); Signalling System No.7; Transaction Capabilities (TC) version 2; Part 1: Protocol specification

[ITU-T Recommendations Q.771 to Q.775 (1993), modified]

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

Foreword

This final draft second edition European Telecommunication Standard (ETS) has been produced by the Signalling Protocols and Switching (SPS) Technical Committee of the European Telecommunications Standards Institute (ETSI), and is now submitted for the Voting phase of the ETSI standards approval procedure.

The second edition of ETS 300 287 covering the Signalling System No.7 Transaction Capabilities (TC) version 2 is structured as a multi-part standard (of which this ETS forms part 1) as described below:

Part 1: "Protocol specification [ITU-T Recommendations Q.771 to Q.775 (1993), modified]";

Part 2: "Protocol Implementation Conformance Statement (PICS) proforma specification";

Part 3: "Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma specification".

Proposed transposition dates	
Date of latest announcement of this ETS (doa):	3 months after ETSI publication
Date of latest publication of new National Standard or endorsement of this ETS (dop/e):	6 months after doa
Date of withdrawal of any conflicting National Standard (dow):	6 months after doa

Endorsement notice

The text of ITU-T Recommendations Q.771, Q.772, Q.773, Q.774 and Q.775 (1993) was approved by ETSI as an ETS with agreed modifications as given below.

NOTE: New or modified text is indicated using sidebars. In addition, underlining and/or strike-out are used to highlight detailed modifications where necessary.

Global modifications to ITU-T Recommendations Q.771 to Q.775

Insert the following two clauses (scope and normative references):

Scope

This first part of ETS 300 287 defines the Transaction Capabilities[1)] (TC) signalling protocol to be used in and between networks, for non-circuit related services which use Signalling System No.7 for inter-network dialogues. Only those parts of TC which are used by the above services need to be provided.

The support of TC by terminal equipment is outside the scope of this ETS.

This ETS covers only TC for use over a network layer consisting of Signalling System No.7 Message Transfer Part (MTP) plus Signalling Connection Control Part (SCCP).

NOTE: This is valid for both 1988 and 1993 versions.

Normative references

This ETS incorporates by dated and undated reference, 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 this ETS only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred to applies.

[1] CCITT Recommendation X.219 (1988): "Information Technology - Open Systems Interconnection - Association Control Service Element (ACSE)".

[2] ITU-T Recommendation X.680 (1994): "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation".

[3] ITU-T Recommendation X.880 (1994): "Information Technology - Remote Operations - Concept, Model and Notation".

[4] ITU-T Recommendation X.881 (1994): "Information Technology - OSI Realisations: Remote Operations Service Element (ROSE) Service Definition".

Services assumed from a connectionless network layer

The services assumed from the SCCP shall be provided via the N-UNITDATA and N-NOTICE primitives.

---

1) For historical reasons, the terms Transaction Capabilities (TC) and Transaction Capabilities Application Part (TCAP) are used interchangeably.

Modifications to ITU-T Recommendation Q.771

Page 9, subclause 3.1.2.1, first bullet item

Replace the definition of "Abort Reason" with the following:

- **"Abort Reason" -** Indicates whether a dialogue is aborted because the received application context name is not supported and no alternative one can be proposed (abort reason = application context not supported), because the dialogue cannot be established for any other user reason (abort reason = dialogue refused) or because a user abort situation has been encountered (abort reason = user specific).

Modifications to ITU-T Recommendation Q.772

Pages 4 and 5, subclauses 3.7.2.3, 3.7.3.3 and 3.7.4.5 "mistyped parameter"

The definition of "mistyped parameter" provided in subclauses 3.7.2.3, 3.7.3.3 and 3.7.4.5 shall also cover error conditions such as ENUMERATED error, value range error, size constraint error, value constraint error and presence constraint error.

Page 7, subclause 4.2.5 "result source diagnostic"

Modify the first sentence of the last paragraph as follows:

The "dialogue service user" can further qualify the result with a diagnostic with values "null" or "no reason given" (for the case where no diagnostic is offered) or "application-context-name-not-supported" (for the case when the dialogue is refused because the application context is not supported).

Modifications to ITU-T Recommendation Q.773

Page 1, subclause 3.1

NOTE: This correction does not change the transfer syntax.

Replace the EXPORTS statement by:

```
EXPORTS OPERATION, ERROR, MessageType, Component, InvokeIdType;
```

Pages 2 and 4, subclause 3.1

NOTE: This correction does not change the transfer syntax.

Add a range constraint of "(0..127)" to all unconstrained INTEGERs, i.e., the following shall be modified:

```
P-AbortCause
GeneralProblem
InvokeProblem
ReturnResultProblem
ReturnErrorProblem
```

When a value, which is not assigned but is within the specified range, is received, this value should be ignored. Values out of range may lead to a syntax error.

Pages 4 and 5, subclause 3.2

NOTE: This correction does not change the transfer syntax.

Add a size constraint of "(1..10)" to "user-information" when carried within AARQ-apdu, AARE-apdu, RLRQ-apdu, RLRE-apdu and ABRT-apdu:

```
user-information    [30] IMPLICIT SEQUENCE SIZE (1..10) OF EXTERNAL OPTIONAL
```

Page 5, subclause 3.2

NOTE: This correction does not change the transfer syntax.

Add a range constraint of "(0..127)" to all unconstrained INTEGERs, i.e., the following shall be modified:

```
ABRT-source
Associate-result
dialogue-service-user
dialogue-service-provider
Release-request-reason
Release-response-reason
```

When a value, which is not assigned but is within the specified range, is received, this value should be ignored. Values out of range may lead to a syntax error.

Page 29, subclause 4.2.3.1, table 46/Q.773

The Result Tag shall be coded 1010 0010.

Page 32, subclause 4.2.3.1, tables 55/Q.773 and 56/Q.773

The integer element shall be “mandatory”.

Page 32, subclause 4.2.3.1, table 57/Q.773

The Dialogue Service Provider Tag shall be coded 1010 0010.

The Dialogue Service User Tag shall be coded 1010 0001.

Modifications to ITU-T Recommendation Q.774

Page 10, table 3/Q.774, note 3

Insert at the end of note 3:

or "dialogue-refused".

Page 12, subclause 3.2.1.2, "Dialogue End"

Replace the penultimate paragraph "If the ... described in 3.2.2" by:

When a TC-User has received a TC-BEGIN indication primitive including some user information it finds unacceptable, it may issue a TC-U-ABORT request primitive with the "Abort Reason" parameter set to "dialogue-refused". This causes a Dialogue Response (AARE) APDU to be formatted. The setting of the values for various fields in the AARE APDU are as follows: the "application-context-name" is identical to the received one, the result field is set to "reject-permanent", and the "result-source-diagnostic" is "dialogue-service-user (null)" or "dialogue-service-user (no-reason-given)".

If the "Abort reason" parameter in the TC-U-ABORT request primitive is absent or has a value other than "application-context-name-not-supported" or "dialogue-refused", this describes an abnormal termination of the dialogue and is described in subclause 3.2.2.

Page 41, figure A.5 (sheet 6 of 11)

Insert in the second decision box before the question mark:

or "dialogue-refused"

Modifications to ITU-T Recommendation Q.775

No modifications identified.

Annex ZA (informative): Realizing the X.880 generic ROS model using TC

ZA.1 Introduction

ZA.1.1 Overview

ITU-T Recommendation X.880 [3] defines a generic model for interactive communication between objects, where the basic interaction involves the invocation of an operation by one object (the invoker) and its performance by another (the performer). This model comes with a set of ASN.1 information object classes to be used by protocol designers in the specification of ROS-based applications.

ITU-T Recommendation X.880 [3] recognizes that there are multiple possible realizations of this model, as far as communication is concerned. The purpose of this annex is to demonstrate that TC can be considered as one of these realizations, by providing a mapping of the generic concepts onto TC services.

ZA.1.2 Notation and concept for the generic ROS model

ITU-T Recommendation X.880 [3] defines several information object classes that are useful in the specification of ROS-based application protocols. Such classes can be used by designers of TC-User applications, as an alternative to the methodology described in ITU-T Recommendation Q.775. These object classes are defined using the information object specification ASN.1 notation defined in ITU-T Recommendation X.881 [4].

The OPERATION class is used to define an operation. It is equivalent to the OPERATION MACRO defined in CCITT Recommendation X.219 [1] and ITU-T Recommendation Q.773 as modified by this ETS.

The ERROR class is used to define an error. It is equivalent to the ERROR MACRO defined in CCITT Recommendation X.219 [1] and ITU-T Recommendation Q.773 as modified by this ETS.

The OPERATION-PACKAGE class is used to define a set of operations which may only be invoked by a ROS-object assuming the role of "consumer", the operations which may only be invoked by a ROS-object assuming the role of "supplier", and the operations which may be invoked by both ROS-objects. When using the communication services of SS7 or OSI, an operation package is realized as an Application Service Element (ASE).

The CONNECTION-PACKAGE class is used to define the bind and unbind operations used to establish and release an association. When realized using the communication services of SS7, a connection package is realized as the procedures that use the structured dialogue handling services of TC. Application contexts which do not require the explicit invocation of bind and unbind operations can still be considered as including a connection package which uses the emptyBind and emptyUnbind pre-defined operations.

The CONTRACT class is used to define an association contract in terms of a connection package and one or more operation packages. When specifying the contract, those packages in which either only the association initiator assumes the role of consumer, or only the association responder assumes the role of consumer, or either may assume the role of consumer, are identified. When using the communication services of SS7 or OSI, a contract is realized as an application context.

The ROS-OBJECT-CLASS class is used to define a set of common capabilities of a set of ROS-objects in terms of the (association) contracts they support as initiators and/or responders. When realized using TC or OSI, a ROS-object maps to an application process and a contract to an application context.

ZA.1.3 Communication model

The realization of ROS involves the selection of a suitable medium to convey invocations and replies between a pair of ROS-objects.

The possible media can be classified in two broad categories:

a) those required when the invoker and the performer are to be implemented in a single physical equipment;

b) those required when the invoker and the performer are to be implemented in separated physical equipment.

Category a) can be further divided into message-passing and procedure-calling facilities.

The medium in category b) depends on the type of network which interconnects the two objects and on some Quality of Service (QoS) criteria.

ITU-T Recommendation X.880 [3] models the medium as being composed of two stub objects (one for the invoker, one for the performer) and one information transfer object (see figure ZA.1). The information transfer object capabilities also includes the association control functionalities which might be required to set up an association between the application entities involved in the communication.

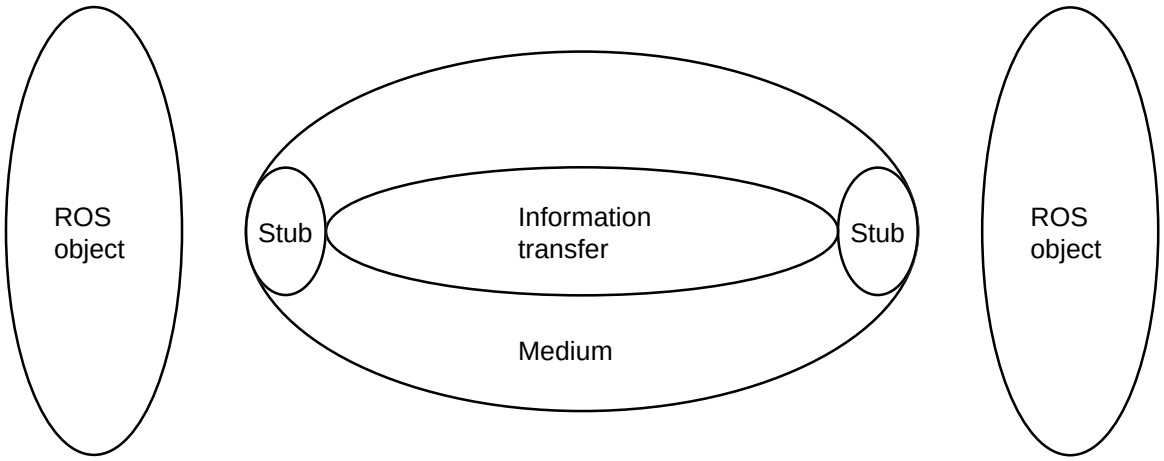

Figure ZA.1: Generic ROS communication model

The role of each stub object is merely to transform invocations and replies into protocol data units (and vice-versa) they exchange using the information transfer object. For a given type of stub objects there are several possible types of information transfer objects.

In the context of OSI, the stub objects are realized by the Remote Operation Service Element (ROSE) while several information transfer realizations are available, using suitable combination of Association Control Service Element (ACSE), Reliable Transfer Service Element (RTSE) and the presentation service.

The stub objects are realized by the Component Handling Block (CHA) of the TC Component Sub-Layer (CSL, see ITU-T Recommendation Q.774 as modified by this ETS) together with a collection of operation-specific ASEs (the TC-User ASEs). The CHA whose services are defined in subclause 3.1.3 of ITU-T Recommendation Q.771 as modified by this ETS drives the generic protocol required to invoke and report returns of arbitrary operations.

Each TC-User ASE embodies knowledge of the definitions of the specific operations involved in some operation package. Collectively, the CSL and the TC-User ASEs have knowledge of all the operations of the association contract.

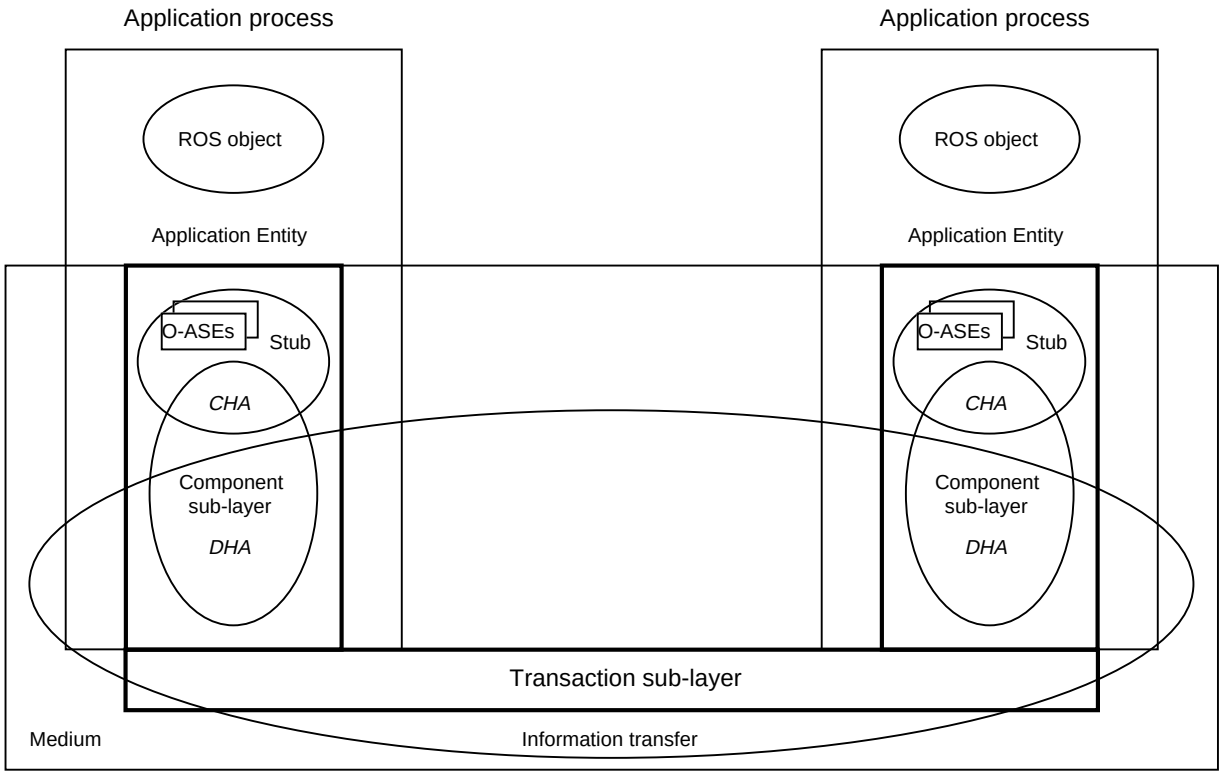

Figure ZA.2: TC realization of ROS

ZA.2 Remote operation service realization

ZA.2.1 Basic services (Stub)

The TC CSL provides the necessary services for supporting the invocation of operations and reporting responses. It also provides additional local services for cancelling operation (TC-U-CANCEL request, TC-L-CANCEL indication) or reporting locally detected protocol error (TC-L-REJECT indication).

NOTE: The following restrictions apply:
- whether an operation is synchronous or not is not taken into account (from a TC point of view operations are always considered as being asynchronous. However, the TC-User might behave in a synchronous manner);
- the set of allowed InvokeIds is restricted to the integer range (-128 to 127);
- the priority field is ignored[2)].

ZA.2.2 Bind and unbind operations

NOTE: In order to minimize the impact of Bind and Unbind operations on TC specifications, this annex assumes that the TC-User constructs the bind and unbind APDUs and transfers them to TC as if it would be ordinary user information. As a consequence, TC is not aware that these operations are being invoked and cannot check that they are used consistently with respect to the dialogue service and component handling service (e.g. it cannot verify that no operation is requested after an unbind operation has been invoked).

---

2) This might evolve as the studies on priority handling in SS7 will progress.

ZA.2.2.1 Bind operation

When an application context definition includes a connection package, the initiating TC-User invokes a bind operation to be executed as part of the dialogue establishment procedure, prior to the execution of any other operation. Failure of the execution of this operation leads to the rejection of the dialogue.

If the TC-User does not really need to invoke an explicit bind operation, it is assumed that it uses the emptyBind pre-defined operation.

ZA.2.2.1.1 Invoking a bind operation

The TC-User can invoke a bind operation using the TC-BEGIN request primitive. If the definition of the bind operation includes an &ArgumentType field, the TC-User constructs a bind-invoke PDU from this information and transfers it as the first (or only part) of the user-information parameter of the TC-BEGIN request primitive. Otherwise no bind-invoke PDU is sent.

NOTE: This should ensure that the bind-request PDU will be included in the first external field of the user-information element of the Dialogue Request APDU (AARQ).

ZA.2.2.1.2 Responding to a bind operation

The TC-User reports the outcome a bind operation using the first dialogue handling primitive it issues.

Successful execution of the bind operation is reported using a TC-CONTINUE request primitive or a TC-END request primitive if there is no need to continue the dialogue. In the latter case it needs also to invoke an unbind operation.

NOTE: Use of the TC-END request primitive at this stage places restriction of the use of unbind operations. It implies that only the responder can unbind and that the unbind operation definition does not include an &ResultType field and that the definition of its associated error does not include an &ParameterType field (e.g. as the emptyUnbind operation).

If the bind operation definition includes an &ResultType field, the TC-User constructs a bind-result PDU from this information and transfers it as the first (or only part) of the user-information parameter of the TC-CONTINUE request primitive or TC-END request primitive. Otherwise no bind-result PDU is sent.

The TC-User reports unsuccessful execution of a bind operation using a TC-U-ABORT request primitive issued as an immediate response to the TC-BEGIN indication. The abort reason parameter takes the value "dialogue-refused".

If the definition of the associated error includes an &ParameterType field, the TC-User constructs a bind-error PDU from this information and transfers it as the first (or only part) of the user-information parameter of the TC-CONTINUE request primitive or TC-END request primitive. Otherwise no bind-error PDU is sent.

The emptyBind operation and the bind-invoke, bind-result and bind-error PDUs are defined in ITU-T Recommendation X.880 [3]. For convenience, their ASN.1 definitions are reproduced below:

```
Bind {OPERATION:operation}   ::= CHOICE
{
    bind-invoke     [16] OPERATION.&ArgumentType (operation),
    bind-result     [17] OPERATION.&ResultType (operation),
    bind-error      [18] OPERATION.&Errors.&ParameterType (operation)
}

emptyBind OPERATION ::= {ERRORS {refuse} SYNCHRONOUS TRUE}
```

Where *operation* refers to the bind operation.

ZA.2.2.2 Unbind operation

ZA.2.2.1.1 Invoking an unbind operation

If the application-context definition includes a connection package, the TC-User invokes an unbind operation as part of the dialogue termination procedure.

The mapping on to TC services depends on the type of this unbind operation:

a) if the unbind operation definition does not include an &ResultType field and the definition of its associated error does not include an &ParameterType field, the operation can be invoked using the TC-END request primitive;

b) if the unbind operation definition includes an &ResultType field or the definition of its associated error includes an &ParameterType field, the operation needs to be invoked using the last TC-CONTINUE request primitive issued by the unbind requestor.

In both cases, if the unbind operation definition includes an &ArgumentType field, the TC-User constructs an unbind-request APDU which is transferred as the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-request APDU is sent.

ZA.2.2.1.2 Responding to an unbind operation

When accepting an unbind operation, the TC-User issues a TC-END request primitive. If the unbind operation definition includes an &ResultType field, the TC-User constructs an unbind-result APDU which is transferred in the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-result APDU is sent.

When refusing an unbind operation, the TC-User issues a TC-CONTINUE request primitive. If the definition of the associated error includes an &ParameterType field, the TC-User constructs an unbind-error APDU which is transferred in the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-result APDU is sent.

NOTE: This should ensure that the unbind-result PDU will be included in the last external field of the user-information element of the Dialogue Response APDU (AARE) when the TC-END request primitive is issued as an immediate response to the TC-BEGIN indication primitive, or otherwise in the single EXTERNAL field of the DialoguePortion.

If the association contract includes a connection package but the TC-User does not need to explicitly invoke an unbind operation, it is assumed that the emptyUnbind operation is used. This operation is conceptually mapped onto the TC-END request primitive, however, no unbind PDU is sent.

The emptyUnbind operation, the unbind-invoke, unbind-result and unbind-error PDUs are defined in ITU-T Recommendation X.880 [3]. For convenience, they are reproduced below:

```
Unbind {OPERATION:operation} ::= CHOICE
{
    unbind-invoke   [19] OPERATION.&ArgumentType (operation),
    unbind-result   [20] OPERATION.&ResultType (operation),
    unbind-error    [21] OPERATION.&Errors.&ParameterType (operation)
}

emptyUnbind OPERATION ::= {SYNCHRONOUS TRUE}
```

Where *operation* refers to the unbind operation.

ZA.3 Information transfer

ZA.3.1 Association realizations

TC provides two association realizations through its dialogue handling function: the structured mode and the unstructured mode which are defined in ITU-T Recommendation Q.771 as modified by this ETS.

ZA.3.2 Transfer realization

As far as Remote Operations are concerned, TC provides the following information transfer capabilities to its user:

- bind and unbind PDUs are transferred in as user-information in the DialoguePortion;

- basic ROS PDUs (plus the return result not last) are transferred in the component portion of any message.

TC provides only one type of transfer realization, irrespective of the type of association realization chosen. However, from a sender's point of view, this realization offers some flexibility to the TC-Users as far as PDU concatenation is concerned.

The Remote Operations TC also provides means to transfer any kind of user information through the use of dialogue handling service primitives.

ZA.4 ROS-based application context

The static aspects of a TC-based application context definition realizing some particular association contract can be described as an information object of class APPLICATION-CONTEXT, which is specified as follows:

```
APPLICATION-CONTEXT ::= CLASS
{
    &associationContract                    CONTRACT,
    &dialogueMode                           DialogueMode,
    &termination                            Termination OPTIONAL,
    &componentGrouping                      BOOLEAN DEFAULT TRUE,
    &dialogueAndComponentGrouping           BOOLEAN DEFAULT TRUE,
    &AbstractSyntaxes                       ABSTRACT-SYNTAX,
    &applicationContextName                 OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX
{
    CONTRACT                                &associationContract
    DIALOGUE MODE                           &dialogueMode
    [TERMINATION                            &termination]
    [COMPONENT GROUPING ALLOWED             &componentGrouping]
    [DIALOGUE WITH COMPONENTS ALLOWED       &dialogueAndComponentGrouping]
    ABSTRACT SYNTAXES                       &AbstractSyntaxes
    APPLICATION CONTEXT NAME                &applicationContextName
}

DialogueMode    ::= ENUMERATED {structured(1), unstructured(2)}

Termination     ::= ENUMERATED {basic(1), pre-arranged(2)}
```

The &associationContract field identifies the association contract which this application context realizes.

The &dialogueMode field indicates whether this application context makes use of the dialogue structured mode facilities or the dialogue unstructured mode facilities. If the association contract definition includes a connection package, the &dialogueMode field indicates "structured".

The &termination field indicates whether basic or pre-arranged termination is used to end the dialogue. If this field is absent, the application-context definition does not place any constraint on which end method is used.

The &componentGrouping field indicates whether components might be grouped in a single message. If this field is absent, the application-context definition does not place any restrictions on this issue.

The &dialogueAndComponentGrouping field indicates whether bind and unbind PDUs can be sent in messages which also contain components. If this field is absent, the application-context definition does not place any restrictions on this issue.

The &AbstractSyntaxes filed contains the abstract syntaxes which are required for the conveyance of information between the objects, including the PDUs for invoking and reporting on the operations in the contract.

The &applicationContextName field contains the value which needs to be provided to TC to identify the application context.

ZA.5 Abstract syntaxes

ZA.5.1 Dialogue control

The &AbstractSyntaxes field of an application context definition needs to include the following abstract syntax if the &dialogueMode field indicates "structured".

```
dialogue-abstract-syntax ABSTRACT-SYNTAX ::=
{
    DialoguePDU IDENTIFIED BY dialogue-as-id
}
```

The &AbstractSyntaxes field of an application context definition needs to include the following abstract syntax if the &dialogueMode field indicates "unstructured".

```
uniDialogue-abstract-syntax ABSTRACT-SYNTAX ::=
{
    UniDialoguePDU IDENTIFIED BY uniDialogue-as-id
}
```

ZA.5.2 User defined syntaxes

ZA.5.2.1 General

The &AbstractSyntaxes field of an application-context definition needs to include one or more abstract syntaxes to represent the TC messages (including the components) and the bind and unbind PDUs. Such abstract syntaxes need to be defined by the application designer.

TC messages are defined in ITU-T Recommendation Q.773 as modified by this ETS, while bind and unbind PDUs are defined in ITU-T Recommendation X.880 [3].

How many abstract syntaxes are defined to support a particular application context is up to the application designer. However, the following rules should be followed.

a) If the application context realizes an association contract which includes a connection package, the values of the data types:

```
Bind    {ac.&associationContract.&connection.&bind}
Unbind  {ac.&associationContract.&connection.&unbind}
```

need to appear in at least one of these abstract syntaxes.

b) For each operation *op* involved in the set of operation packages used by the application context, there need to be at least one of the abstract syntaxes which include the values of the following types:

```
Invoke          {TCInvokeIds, OPERATION: op}
ReturnResult    {OPERATION: op}
```

c) For each error *err* involved in the set of operation packages used by the application context, there need to be at least one of the abstract syntaxes which include the values of the following types:

```
ReturnError {ERROR: err}
```

d) At least one of the abstract syntaxes needs to include:

```
Reject
```

ZA.5.2.2 Defining the abstract syntaxes

Given an operation package, a single abstract syntax which allows the exchange of TC messages carrying invocation and reporting for all of its operations can be defined using the following data type:

```
TCSingleAS {OPERATION-PACKAGE: package} ::=
TCMessage {{AllOperations {package}}, {AllOperations {package}}}
```

Or, alternatively, a pair of abstract syntaxes can be defined based upon the pair of types:

```
TCConsumerAS {OPERATION-PACKAGE: package} ::=
TCMessage {{ConsumerPerforms {package}}, {ConsumerPerforms {package}}}

TCSupplierAS{OPERATION-PACKAGE: package} ::=
TCMessage {{SupplierPerforms {package}}, {SupplierPerforms {package}}}
```

A single abstract syntax may accommodate a set of packages, provided that the operation and error codes are unique. For example, the following data type can be used as the basis of a single abstract syntax to accommodate all the operation packages involved in an association contract:

```
AllPackagesAS {APPLICATION-CONTEXT: ac} ::=
    TCSingleAS
        {
        combine
            {
                {
                  ac.&associationContract.&OperationsOf
                | ac.&associationContract.&InitiatorConsumerOf
                | ac.&associationContract.&InitiatorSupplierOf
                },
                {},
                {}
            }
        }
```

An independent abstract syntax can be defined to represent values of bind and unbind PDUs, based on the following type:

```
ConnectionAS {APPLICATION-CONTEXT: ac} ::= CHOICE
{
    bind        Bind    {ac.&associationContract.&connection.&bind},
    unbind      Unbind  {ac.&associationContract.&connection.&unbind}
}
```

The value of the corresponding &abstract-syntax-name field is intended to serve as a direct reference when values of these PDUs are conveyed in the user-information parameter of Dialogue Control PDUs, or directly as value of the DialoguePortion.

ZA.6 ASN.1 module

```
TC-Notation-Extensions {ccitt recommendation q 775 modules(2) notation-extension(4) version 1(1)}

DEFINITIONS ::=

BEGIN

IMPORTS

TCMessage{}
    FROM TCAPMessages
    {ccitt recommendation q 773 modules(2) messages(1) version3(3)}

Bind, Unbind
    FROM Remote-Operations-Generic-ROS-PDUs
    {joint-iso-ccitt remote-operations(4) generic-ROS-PDUs(6) version1(0)}

combine{}, AllOperations{}, ConsumerPerforms{}, SupplierPerforms{}
    FROM Remote-Operations-Useful-Definitions
    {joint-iso-ccitt remote-operations(4) useful-definitions(7) version1(0)}

CONTRACT, OPERATION-PACKAGE
    FROM Remote-Operations-Information-Objects
    {joint-iso-ccitt remote-operations(4) informationObjects(5) version1(0)}
```

```
UniDialoguePDU, uniDialogue-as-id
    FROM UnidialoguePDUs
    {ccitt recommendation q 773 modules(2) unidialoguePDUs(3) version1(1)}

DialoguePDU, dialogue-as-id
    FROM DialoguePDUs
    {ccitt recommendation q 773 modules(2) dialoguePDUs(2) version1(1)}
;

APPLICATION-CONTEXT ::= CLASS
{
    &associationContract                    CONTRACT,
    &dialogueMode                           DialogueMode,
    &termination                            Termination OPTIONAL,
    &componentGrouping                      BOOLEAN DEFAULT TRUE,
    &dialogueAndComponentGrouping           BOOLEAN DEFAULT TRUE,
    &AbstractSyntaxes                       ABSTRACT-SYNTAX,
    &applicationContextName                 OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX
{
    CONTRACT                                &associationContract
    DIALOGUE MODE                           &dialogueMode
    [TERMINATION                            &termination]
    [COMPONENT GROUPING ALLOWED             &componentGrouping]
    [DIALOGUE WITH COMPONENTS ALLOWED       &dialogueAndComponentGrouping]
    ABSTRACT SYNTAXES                       &AbstractSyntaxes
    APPLICATION CONTEXT NAME                &applicationContextName
}

DialogueMode    ::= ENUMERATED {structured(1), unstructured(2)}

Termination     ::= ENUMERATED {basic(1), pre-arranged(2)}

dialogue-abstract-syntax ABSTRACT-SYNTAX ::=
{
    DialoguePDU IDENTIFIED BY dialogue-as-id
}

uniDialogue-abstract-syntax ABSTRACT-SYNTAX ::=
{
    UniDialoguePDU IDENTIFIED BY uniDialogue-as-id
}

TCSingleAS {OPERATION-PACKAGE: package} ::=
TCMessage {{AllOperations {package}}, {AllOperations {package}}}

TCConsumerAS {OPERATION-PACKAGE: package} ::=
TCMessage {{ConsumerPerforms {package}}, {ConsumerPerforms {package}}}

TCSupplierAS{OPERATION-PACKAGE: package} ::=
TCMessage {{SupplierPerforms {package}}, {SupplierPerforms {package}}}

AllPackagesAS {APPLICATION-CONTEXT: ac} ::=
    TCSingleAS
        {
        combine
            {
                {
                  ac.&associationContract.&OperationsOf
                | ac.&associationContract.&InitiatorConsumerOf
                | ac.&associationContract.&InitiatorSupplierOf
                },
                {},
                {}
            }
        }

ConnectionAS {APPLICATION-CONTEXT: ac} ::= CHOICE
{
    bind        Bind    {ac.&associationContract.&connection.&bind},
    unbind      Unbind  {ac.&associationContract.&connection.&unbind}
}


END -- TC-Notation-Extensions
```

Annex ZB (informative): TCAPMessage module using 1994 ASN.1 notation

The following module has been written using the version of ASN.1 as defined in ITU-T Recommendation X.680 [2]. This module defines the parameterized type TCMessage which provides a basis for the definition of an abstract syntax containing TCAP messages. This type is parameterized in that the values conveyed in the component portion depend on the set of operations which can be invoked and the set of operations for which a response may be generated.

```
TCAPMessages {ccitt recommendation q 773 modules(2) messages(1) version3(3)}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN
-- EXPORTS everything

-- Transaction Portion fields.

IMPORTS

ROS{}
    FROM Remote-Operations-Generic-ROS-PDUs
    {joint-iso-ccitt remote-operations(4) generic-ROS-PDUs(6) version1(0)}

OPERATION
    FROM Remote-Operations-Information-Objects
    {joint-iso-ccitt remote-operations(4) informationObjects(5) version1(0)}
;

TCMessage {OPERATION: Invokable, OPERATION: Returnable} ::= CHOICE
{
    unidirectional  [APPLICATION 1] Unidirectional {{Invokable}, {Returnable}},
    begin           [APPLICATION 2] Begin {{Invokable}, {Returnable}},
    end             [APPLICATION 4] End {{Invokable}, {Returnable}},
    continue        [APPLICATION 5] Continue {{Invokable}, {Returnable}},
    abort           [APPLICATION 7] Abort
}

Unidirectional {OPERATION: Invokable, OPERATION: Returnable} ::=     SEQUENCE
{
    dialoguePortion     DialoguePortion     OPTIONAL,
    components          ComponentPortion    {{Invokable}, {Returnable}}
}

Begin {OPERATION: Invokable, OPERATION: Returnable} ::= SEQUENCE
{
    otid                OrigTransactionID,
    dialoguePortion     DialoguePortion     OPTIONAL,
    components          ComponentPortion    {{Invokable}, {Returnable}} OPTIONAL
}

End {OPERATION: Invokable, OPERATION: Returnable}::= SEQUENCE
{
    dtid                DestTransactionID,
    dialoguePortion     DialoguePortion     OPTIONAL,
    components          ComponentPortion    {{Invokable}, {Returnable}} OPTIONAL
}

Continue {OPERATION: Invokable, OPERATION: Returnable} ::=   SEQUENCE
{
    otid                OrigTransactionID,
    dtid                DestTransactionID,
    dialoguePortion     DialoguePortion     OPTIONAL,
    components          ComponentPortion    {{Invokable}, {Returnable}} OPTIONAL
}

Abort ::= SEQUENCE
{
    dtid                DestTransactionID,
    reason CHOICE
    {
        p-abortCause        P-AbortCause,
        dialoguePortion     DialoguePortion
    } OPTIONAL
}

-- NOTE:    When the Abort Message is generated by the Transaction sublayer, a p-Abort Cause
--          needs to be present.
```

```
DialoguePortion  ::= [APPLICATION 11] EXPLICIT EXTERNAL

-- The dialogue portion carries the dialogue control PDUs as value of the external data type. The
-- direct reference should be set to:
--      {ccitt recommendation q 773 as(1) dialogue-as(1) version1(1)}
-- if structured dialogue is used,
-- and to:
--      {ccitt recommendation q 773 as(1) unidialogue-as(2) version1(1)}
-- if unstructured dialogue is used.

OrigTransactionID   ::= [APPLICATION 8] OCTET STRING(SIZE(1..4))

DestTransactionID   ::= [APPLICATION 9] OCTET STRING(SIZE(1..4))

P-AbortCause        ::= [APPLICATION 10] INTEGER {
    unrecognizedMessageType             (0),
    unrecognizedTransactionID           (1),
    badlyFormattedTransactionPortion    (2),
    incorrectTransactionPortion         (3),
    resourceLimitation                  (4)
} (0..127)

-- COMPONENT PORTION. The last field in the transaction portion of the TCAP message is the
-- component portion. The component portion may be absent.

ComponentPortion {OPERATION: Invokable, OPERATION: Returnable}   ::=
    [APPLICATION 12] SEQUENCE SIZE (1..MAX) OF Component {{Invokable}, {Returnable}}

-- Component Portion fields

-- COMPONENT TYPE. Recommendation X.880 defines four APDUs for invoking operations, returning
-- results or error, and for the rejection of invalid PDUs. TCAP adds returnResultNotLast to
-- allow for the segmentation of a result.

Component {OPERATION: Invokable, OPERATION: Returnable} ::= CHOICE
{
    basicROS                                    ROS {TCInvokeIdSet, {Invokable}, {Returnable}},
    returnResultNotLast     [7] returnResult < ROS {TCInvokeIdSet, {Invokable}, {Returnable}}
}

TCInvokeIdSet INTEGER ::= {-128..127}


END -- TCAPMessages
```

NOTE 1: The parameterized type ROS{} defined in ITU-T Recommendation X.880 [3] represents the four basic ROS PDUs: invoke, return result, return error and reject.

NOTE 2: The information object class OPERATION defined in ITU-T Recommendation X.880 [3] is the replacement for the OPERATION MACRO.

NOTE 3: Invokable and Returnable are two set of operations.

History

Document history			
October 1993	First Edition of ETS 300 287		
September 1995	Public Enquiry	PE 92:	1995-09-25 to 1996-01-19
September 1996	Vote	V 110:	1996-09-09 to 1996-11-01