INTERNATIONAL TELECOMMUNICATION UNION

CCITT X.711

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

DATA COMMUNICATION NETWORKS: OPEN SYSTEMS INTERCONNECTION (OSI); MANAGEMENT

COMMON MANAGEMENT INFORMATION PROTOCOL SPECIFICATION FOR CCITT APPLICATIONS

Recommendation X.711

Geneva, 1991

FOREWORD

The CCITT (the International Telegraph and Telephone Consultative Committee) is the permanent organ of the International Telecommunication Union (ITU). CCITT is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The Plenary Assembly of CCITT which meets every four years, establishes the topics for study and approves Recommendations prepared by its Study Groups. The approval of Recommendations by the members of CCITT between Plenary Assemblies is covered by the procedure laid down in CCITT Resolution No. 2 (Melbourne, 1988).

Recommendation X.711 was prepared by Study Group VII and was approved under the Resolution No. 2 procedure on the 22nd of March 1991.

CCITT NOTE

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

© ITU 1991

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the ITU.

Recommendation X.711

COMMON MANAGEMENT INFORMATION PROTOCOL SPECIFICATION FOR CCITT APPLICATIONS[1)]

CONTENTS

1 *Scope*

2 *References*

3 *Definitions*

4 *Symbols and abbreviations*

5 *Overview*

6 *Elements of procedure*

7 *Abstract syntax*

8 *Conformance*

Annex A – Association rules for CMISE

Annex B – Expanded ASN.1 syntax

Annex C – Examples of CMISE ROSE APDUs

1 Scope

This Recommendation specifies a protocol which is used by application layer entities to exchange management information.

This Recommendation specifies:

- procedures for the transmission of management information between application entities;
- the abstract syntax of the Common Management Information Protocol (CMIP) and the associated encoding rules to be applied;
- procedures for the correct interpretation of protocol control information;
- the conformance requirements to be met by implementation of this Recommendation.

This Recommendation does not specify:

- the structure or meaning of the management information that is transmitted by means of CMIP;
- the manner in which management is accomplished as a result of CMIP exchanges;
- the interactions which result in the use of CMIP.

---

1) Recommendation X.711 and ISO/IEC 9596, *Information technology – Open systems interconnection – Common management information protocol specification* were developed in close collaboration and are technically identical.

2 References

[1] CCITT Recommendation *Reference Model of Open Systems Interconnection for CCITT Applications*, Blue Book, Fascicle VIII.4, Rec. X.200, ITU, Geneva, 1988.

[2] ISO/IEC 7498-4 – *Information processing systems – Open Systems Interconnection – Basic Reference model – Part 4: Management framework 1989*.

[3] CCITT Recommendation *Specification of abstract syntax notation one (ASN.1)*, Blue Book, Fascicle VIII.4, Rec. X.208, ITU, Geneva, 1988.

[4] CCITT Recommendation *Specification of Basic Encoding Rules for abstract syntax notation one (ASN.1)*, Blue Book, Fascicle VIII.4, Rec. X.209, ITU, Geneva, 1988.

[5] CCITT Recommendation *Session Service Definition for Open Systems Interconnection for CCITT Applications*, Blue Book, Fascicle VIII.4, Rec. X.215, ITU, Geneva, 1988.

[6] CCITT Recommendation *Presentation Service Definition for Open Systems Interconnection for CCITT Applications*, Blue Book, Fascicle VIII.4, Rec. X.216, ITU, Geneva, 1988.

[7] CCITT Recommendation *Association Control Service Definition for Open Systems interconnection for CCITT Applications*, Blue Book, Fascicle VIII.4, Rec. X.217, ITU, Geneva, 1988.

[8] CCITT Recommendation *Remote Operations: Model, Notation and Service Definition*, Blue Book, Fascicle VIII.4, Rec. X.219, ITU, Geneva, 1988.

[9] CCITT Recommendation *Presentation Protocol Specification for Open Systems Interconnection for CCITT Applications*, Blue Book, Fascicle VIII.5, Rec. X.226, ITU, Geneva, 1988.

[10] CCITT Recommendation *Association Control Protocol Specification for Open Systems interconnection for CCITT Applications*, Blue Book, Fascicle VIII.5, Rec. X.227, ITU, Geneva, 1988.

[11] CCITT Recommendation *Remote Operations: Protocol Specification*, Blue Book, Fascicle VIII.5, Rec. X.229, ITU, Geneva, 1988.

[12] CCITT Recommendation *Common Management Information Service Definition for CCITT Applications*, Rec. X.710, ITU, Geneva, 1991.

3 Definitions

For the purposes of this Recommendation, the following definitions apply.

3.1 *Basic Reference Model definitions*

This Recommendation makes use of the following terms defined in Recommendation X.200 [1]:

a) application-service-element;

b) application-process;

c) real open system;

d) systems-management.

3.2 *Management Framework definitions*

This Recommendation makes use of the following terms defined in ISO/IEC 7498-4 [2]:

a) managed object;

b) management information;

c) management information base;

d) systems management application-entity.

3.3 *Remote Operations definitions*

This Recommendation makes use of the following terms defined in Recommendation X.219 [8]:

a) association-initiator;

b) association-responder;

c) linked-operations;

d) Remote Operations;

e) Remote Operation Service Element;

f) invoker;

g) performer;

h) Association Class;

i) Operation Class.

3.4 *CMIS definitions*

This Recommendation makes use of the following terms defined in Recommendation X.710 [12]:

a) attribute;

b) common management information service element;

c) common management information services;

d) CMISE-service-provider;

e) CMISE-service-user;

f) invoking CMISE-service-user;

g) performing CMISE-service-user.

3.5 *ACSE definitions*

This Recommendation makes use of the following terms defined in Recommendation X.217 [7]:

a) application context;

b) application-association;

c) association.

3.6 *Presentation definitions*

This Recommendation makes use of the following terms defined in Recommendation X.216 [6]:

a) abstract syntax;

b) transfer syntax.

4 Symbols and abbreviations

ACSE Association Control Service Element

APDU Application protocol data unit

ASE Application Service Element

ASN.1 Abstract Syntax Notation One

CMIP Common management information protocol

CMIPM Common management information protocol machine

CMIS Common Management Information service

CMISE Common Management Information Service Element

DCS Defined context set

PCI Protocol control information

PDU Protocol data unit

PICS Protocol implementation conformance statement

RO Remote operations

ROSE Remote Operations Service Element

SMAE Systems management application-entity

5 Overview

The common management information protocol (CMIP) specifies protocol elements that may be used to provide the operation and notification services described in Recommendation X.710 [12], which defines the Common Management Information Services (CMIS).

5.1 *Service provided*

The protocol specified in this Recommendation supports the services defined in Recommendation X.710 [12]. These services are summarized in Table 1/X.711.

TABLE 1/X.711

Common management information services

Service	Type
M-CANCEL-GET	confirmed
M-EVENT-REPORT	confirmed/non-confirmed
M-GET	confirmed
M-SET	confirmed/non-confirmed
M-ACTION	confirmed/non-confirmed
M-CREATE	confirmed
M-DELETE	confirmed

5.2 *Underlying services*

This Recommendation uses the RO-INVOKE, RO-RESULT, RO-ERROR and RO-REJECT-U services of the Remote Operations Service Element (ROSE) defined in Recommendation X.219 [8]. ROSE assumes the use of the presentation service defined in Recommendation X.216 [6]. The confirmed operations of CMIP are operation class 2 (asynchronous) or operation class 1 (synchronous) as required by the application. The choice of operation class is a local matter. The unconfirmed operations of CMIP are operation class 5 (asynchronous, outcome not reported). CMIP uses Association class 3.

If the extended service functional unit is successfully negotiated, ROSEapdus may be mapped on to presentation services other than the P-DATA service.

Note – For example, it may be necessary to modify the presentation defined context set (DCS) when the CMIP operation is sent to the peer CMISE-service-user. In this case, the ROSE APDU which carries the CMIP operation will be mapped onto the P-ALTER-CONTEXT service which is also used to perform the changes to the DCS.

Details of which other presentation services are required and how they are used, are described in the description of the application context in use on the association.

5.2.1 *Service assumed from the ACSE*

This Recommendation assumes the use of the A-ASSOCIATE, A-RELEASE, A-ABORT, and A-P-ABORT services of the Association Control Service Element.

5.2.2 *Service assumed from the presentation layer*

Recommendation X.229 [11] assumes the use of the P-DATA service of the presentation layer for the transfer of the RO-INVOKE, RO-RESULT, RO-ERROR and RO-REJECT PDUs.

5.3 *Management information definitions*

This Recommendation defines the abstract syntax of the Common Management Information Protocol. Attributes specific to a particular managed object are specified by the Recommendation which defines that object.

6 Elements of procedure

This clause provides definition for the procedural elements of the CMIP. The procedures define the transfer of CMIP PDUs whose structure, coding and relationship with the CMIS service primitives is specified in § 7.

The Common Management Information Protocol Machine (CMIPM) accepts CMIS request and response service primitives, and issues CMIP PDUs initiating specific elements of procedure as specified in this clause.

A CMIPM shall accept any well-formed CMIP PDU, and pass it to the performing CMISE-service-user for processing, by means of CMIS indication and confirmation service primitives. If the received PDU is not well formed or does not contain a supported notification or operation, a PDU is returned indicating that the received PDU has been rejected.

The procedures indicate only how to interpret the various fields in the CMIP PDU, not what an invoking CMISE-service-user should do with the information it requests nor how a performing CMISE-service-user should process the invocation.

6.1 *Association establishment*

The establishment of an association involves two CMISE-service-users, one that is the association-initiator and one that is the association-responder.

A CMISE-service-user may initiate an association establishment by using the A-ASSOCIATE service of Recommendation X.217 [7].

The application context specifies, among other things, the rules required for the coordination of initialisation information corresponding to different ASEs. The association rules for CMISE are specified in Annex A.

6.2 *Remote operations*

6.2.1 *RO elements of procedure*

The CMIP elements of procedure rely on the following underlying remote operations elements of procedure

a) invocation;

b) return-result;

c) return-error;

d) user-reject;

e) provider-reject.

These elements of procedure are described fully in Recommendation X.229 [11].

Table 2/X.711 specifies the correspondence between CMIS and ROSE parameters.

TABLE 2/X.711

Correspondence between CMIS and ROSE parameters

CMIS parameter	ROSE parameter
Invoke identifier	InvokeID
Linked identifier	Linked-ID

The correspondence between other CMIS and ROSE parameters is specified in § 7.

6.2.2 *RO-Reject problem parameters*

The RO-Reject problem parameters are mapped or processed as follows

6.2.2.1 RO-Reject-User.Invoke-problem mapping to CMIS error codes is specified in Table 3/X.711.

TABLE 3/X.711

Mapping RO-Reject-User. Invoke-problem to CMISE error codes

RO-REJECT parameter	CMISE error code
duplicate-invocation	duplicate invocation
mistyped-argument	mistyped argument
resource-limitation	resource limitation
unrecognized-operation	unrecognized operation

Other Invoke-problem parameters are a local matter.

6.2.2.2 Other RO-Reject parameters will be handled as a local matter.

6.3 *Event reporting procedure*

6.3.1 *Invocation*

The event reporting procedures are initiated by the M-EVENT-REPORT request primitive.

On receipt of the M-EVENT-REPORT request primitive, the CMIPM shall

a) in the confirmed mode, construct an APDU requesting the m-EventReport-Confirmed operation, otherwise, construct an APDU requesting the m-EventReport operation;

b) send the APDU using the RO-INVOKE procedure.

6.3.2 *Receipt*

On receipt of an APDU requesting either the m-EventReport or m-EventReport-Confirmed operation, the CMIPM shall, if the APDU is well formed, issue an M-EVENT-REPORT indication primitive to the CMISE-service-user with the mode parameter indicating whether or not confirmation is requested, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.3.3 *Response*

In the confirmed mode, the CMIPM shall accept an M-EVENT-REPORT response primitive and shall

a) construct an APDU confirming the M-EVENT-REPORT notification;

b) if the parameters in the M-EVENT-REPORT response primitive indicate that the notification was accepted, send the APDU using the RO-RESULT procedure, otherwise, send the APDU using the RO-ERROR procedure.

6.3.4 *Receipt of response*

On receipt of an APDU responding to an M-EVENT-REPORT notification, the CMIPM shall, if the APDU is well formed, issue an M-EVENT-REPORT confirmation primitive to the CMISE-service-user, thus completing the notification procedure, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.4 *Get procedure*

6.4.1 *Invocation*

The Get procedures are initiated by the M-GET request primitive.

On receipt of the M-GET request primitive, the CMIPM shall

a) construct an APDU requesting the m-Get operation;

b) send the APDU using the RO-INVOKE procedure.

6.4.2 *Receipt*

On receipt of an APDU requesting the m-Get operation, the CMIPM shall, if the APDU is well formed, issue an M-GET indication primitive to the CMISE-service-user, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.4.3 *Response*

The CMIPM shall

a) accept zero or more M-GET response primitives containing a linked-ID followed by a single M-GET response primitive without a linked-ID;

b) or each M-GET response primitive containing a linked-ID the CMIPM shall

– construct an APDU requesting the m-Linked-Reply operation with LinkedReplyArgument set appropriately as either getListError, getResult or processingFailure;

– send each APDU using the RO-INVOKE procedure

c) for the M-GET response primitive not containing a linked-ID the CMIPM shall

– construct an APDU confirming the m-Get operation;

– if the parameters in the M-GET response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure. If the parameters in the M-GET response primitive indicate that the operation was performed with partial success or was not performed because of an error, send the APDU using the RO-ERROR procedure.

6.4.4 *Receipt of response*

On receipt of an APDU responding to an m-Get operation, the CMIPM shall

a) if the APDU included a linked-ID and is well formed, issue an M-GET confirm primitive to the CMISE-service-user;

b) if the APDU is the last response (i.e. not containing a linked-ID) and is well formed, issue an M-GET confirmation primitive to the CMISE-service-user, thus completing the M-GET procedure;

c) if the APDU is not well formed, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.4.5 *CancelGet procedure*

6.4.5.1 *Invocation*

The CancelGet procedures are initiated by the M-CANCEL-GET request primitive.

On receipt of the M-CANCEL-GET request primitive, the CMIPM shall

a) construct an APDU requesting the m-CancelGet operation;

b) send the APDU using the RO-INVOKE procedure.

6.4.5.2 *Receipt*

On receipt of an APDU requesting the m-CancelGet operation, the CMIPM shall, if the APDU is well formed, issue an M-CANCEL-GET indication primitive to the CMISE-service-user, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.4.5.3 *Response*

The CMIPM shall:

a) construct an APDU confirming the m-CancelGet operation;

b) if the parameters in the M-CANCEL-GET response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure otherwise, send the APDU using the RO-ERROR procedure. If the M-CANCEL-GET operation is successful, the performing CMISE-service-user shall cease from sending linked replies to the M-GET operation and shall issue an M-GET response primitive which shall contain the “operation cancelled” error.

6.4.5.4 *Receipt of response*

On receipt of an APDU responding to an m-CancelGet operation, the CMIPM shall, if the APDU is well formed, issue an M-CANCEL-GET confirm primitive to the CMISE-service-user, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.5 *Set procedure*

6.5.1 *Invocation*

The Set procedures are initiated by the M-SET request primitive.

On receipt of the M-SET request primitive, the CMIPM shall

a) in the confirmed mode, construct an APDU requesting the m-Set-Confirmed operation, otherwise, construct an APDU requesting the m-Set operation,

b) send the APDU using the RO-INVOKE procedure.

6.5.2 *Receipt*

On receipt of an APDU requesting the m-Set or m-Set-Confirmed operation, the CMIPM shall, if the APDU is well formed, issue an M-SET indication primitive to the CMISE-service-user, with the mode parameter indicating whether or not confirmation is requested, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.5.3 *Response*

In the confirmed mode, the CMIPM shall:

a) accept zero or more M-SET response primitives containing a linked-ID followed by a single M-SET response primitive without a linked-ID;

b) for each M-SET response primitive containing a linked-ID the CMIPM shall

- construct an APDU requesting the m-Linked-Reply operation with LinkedReplyArgument set appropriately as either setListError, setResult or processingFailure;

- send each APDU using the RO-INVOKE procedure;

c) for the M-SET response primitive not containing a linked-ID the CMIPM shall

- construct an APDU confirming the m-Set operation;

- if the parameters in the M-SET response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure. If the parameters in the M-SET response primitive indicate that the operation was performed with partial success or was not performed because of an error, send the APDU using the RO-ERROR procedure.

6.5.4 *Receipt of response*

On receipt of an APDU responding to an m-Set-Confirmed operation, the CMIPM shall

a) if the APDU included a linked-ID and is well formed, issue an M-SET confirm primitive to the CMISE-service-user;

b) if the APDU is the last response (i.e. not containing a linked-ID) and is well formed, issue an M-SET confirmation primitive to the CMISE-service-user, thus completing the M-SET procedure;

c) if the APDU is not well formed, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.6 *Action procedure*

6.6.1 *Invocation*

The Action procedures are initiated by the M-ACTION request primitive.

On receipt of the M-ACTION request primitive, the CMIPM shall

a) in the confirmed mode, construct an APDU requesting the m-Action-Confirmed operation otherwise, construct an APDU requesting the m-Action operation,

b) send the APDU using the RO-INVOKE procedure.

6.6.2 *Receipt*

On receipt of an APDU requesting the m-Action or m-Action-Confirmed operation, the CMIPM shall, if the APDU is well formed, issue an M-ACTION indication primitive to the CMISE-service-user, with the mode parameter indicating whether or not confirmation is requested, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.6.3 *Response*

In the confirmed mode, the CMIPM shall

a) accept zero or more M-ACTION response primitives containing a linked-ID followed by a single M-ACTION response primitive without a linked-ID;

b) for each M-ACTION response primitive containing a linked-ID the CMIPM shall

– construct an APDU requesting the m-Linked-Reply operation with LinkedReplyArgument set appropriately as either actionError, actionResult or processingFailure;

– send each APDU using the RO-INVOKE procedure;

c) for the M-ACTION response primitive not containing a linked-ID the CMIPM shall

– construct an APDU confirming the m-Action operation;

– if the parameters in the M-ACTION response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure, otherwise, send the APDU using the RO-ERROR procedure.

6.6.4 *Receipt of response*

On receipt of an APDU responding to an m-Action-Confirmed operation, the CMIPM shall

a) if the APDU included a linked-ID and is well formed, issue an M-ACTION confirm primitive to the CMISE-service-user;

b) if the APDU is the last response (i.e. not containing a linked-ID) and is well formed, issue an M-ACTION confirmation primitive to the CMISE-service-user, thus completing the M-ACTION procedure;

c) if the APDU is not well formed, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.7 *Create procedure*

6.7.1 *Invocation*

The Create procedures are initiated by the M-CREATE request primitive.

On receipt of the M-CREATE request primitive, the CMIPM shall

a) construct an APDU requesting the m-Create operation,

b) send the APDU using the RO-INVOKE procedure.

6.7.2 *Receipt*

On receipt of an APDU requesting the m-Create operation, the CMIPM shall, if the APDU is well formed, issue an M-CREATE indication primitive to the CMISE-service-user, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.7.3 *Response*

The CMIPM shall accept an M-CREATE response primitive and shall

a) construct an APDU confirming the m-Create operation,

b) if the parameters in the M-CREATE response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure, otherwise, send the APDU using the RO-ERROR procedure.

6.7.4 *Receipt of response*

On receipt of an APDU responding to an m-Create operation, the CMIPM shall, if the APDU is well formed, issue an M-CREATE confirmation primitive to the CMISE-service-user, thus completing the M-CREATE procedure, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.8 *Delete procedure*

6.8.1 *Invocation*

The Delete procedures are initiated by the M-DELETE request primitive.

On receipt of the M-DELETE request primitive, the CMIPM shall

a) construct an APDU requesting the m-Delete operation,

b) send the APDU using the RO-INVOKE procedure.

6.8.2 *Receipt*

On receipt of an APDU requesting the m-Delete operation, the CMIPM shall, if the APDU is well formed, issue an M-DELETE indication primitive to the CMISE-service-user, otherwise, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.8.3 *Response*

The CMIPM shall

a) accept zero or more M-DELETE response primitives containing a linked-ID followed by a single M-DELETE response primitive without a linked-ID;

b) for each M-DELETE response primitive containing a linked-ID the CMIPM shall

- construct an APDU requesting the m-Linked-Reply operation with LinkedReplyArgument set appropriately as either deleteError, deleteResult or processingFailure;
- send each APDU using the RO-INVOKE procedure;

c) for the M-DELETE response primitive not containing a linked-ID the CMIPM shall

- construct an APDU confirming the m-Delete operation;
- if the parameters in the M-DELETE response primitive indicate that the operation was performed correctly, send the APDU using the RO-RESULT procedure, otherwise, send the APDU using the RO-ERROR procedure.

6.8.4 *Receipt of response*

On receipt of an APDU responding to an m-Delete operation, the CMIPM shall

a) if the APDU included a linked-ID and is well formed, issue an M-DELETE confirm primitive to the CMISE-service-user;

b) if the APDU is the last response (i.e. not containing a linked-ID) and is well formed, issue an M-DELETE confirmation primitive to the CMIS-service-user, thus completing the M-DELETE procedure;

c) if the APDU is not well formed, construct an APDU containing notification of the error and send it using the RO-REJECT-U procedure.

6.9 *Association orderly release*

Either CMISE-service-user may initiate an orderly release of the association by using the A-RELEASE service of Recommendation X.217 [7].

Note – This specification is different from the ROSE use of the BIND operation in which only the association-initiator may use the A-RELEASE procedure.

6.10 *Association abrupt release*

Either CMISE-service-user may initiate an abrupt release of the association using the A-ABORT service of Recommendation X.217 [7].

The CMISE-service-provider may initiate an abrupt release of the association using the A-P-ABORT service of Recommendation X.217 [7].

7 Abstract syntax

This clause specifies the abstract syntax for the CMIP PDUs.

7.1 *Conventions*

The abstract syntax is defined using the notation specified in Recommendation X.208 [3]. The ASN.1 MACRO productions used or referenced by this Recommendation do not exercise the ambiguous aspects of the grammar.

For each of the CMISE service parameters which is to be transferred by a CMIP PDU, there is a PDU field (an ASN.1 NamedType) with the same name as the corresponding service parameter (see Recommendation X.710 [12]), except for the differences required by the use of ASN.1, which are that blanks between words are removed and the first letter of the following word is capitalized, e.g. “managed object class” becomes “managedObjectClass”. To make some of the names shorter, some words are abbreviated as follows:

ack acknowledgement

arg argument

id identifier

info information

sync synchronization.

7.2 *Correspondence between CMISE primitives and CMIP operations*

TABLE 4/X.711

Correspondence between CMISE primitives and CMIP operations

CMIS primitive	Mode	Linked-ID	CMIP operation
M-CANCEL-GET req/ind	Confirmed	Not applicable	m-Cancel-Get-Confirmed
M-CANCEL-GET rsp/conf	Not applicable	Not applicable	m-Cancel-Get-Confirmed
M-EVENT-REPORT req/ind	Non-confirmed	Not applicable	m-EventReport
M-EVENT-REPORT req/ind	Confirmed	Not applicable	m-EventReport-Confirmed
M-EVENT-REPORT rsp/conf	Not applicable	Not applicable	m-EventReport-Confirmed
M-GET req/ind	Confirmed	Not applicable	m-Get
M-GET rsp/conf	Not applicable	Absent	m-Get
M-GET rsp/conf	Not applicable	Present	m-Linked-Reply
M-SET req/ind	Non-confirmed	Not applicable	m-Set
M-SET req/ind	Confirmed	Not applicable	m-Set-Confirmed
M-SET rsp/conf	Not applicable	Absent	m-Set-Confirmed
M-SET rsp/conf	Not applicable	Present	m-Linked-Reply
M-ACTION req/ind	Non-confirmed	Not applicable	m-Action
M-ACTION req/ind	Confirmed	Not applicable	m-Action-confirmed
M-ACTION rsp/conf	Not applicable	Absent	m-Action-confirmed
M-ACTION rsp/conf	Not applicable	Present	m-Linked-Reply
M-CREATE req/ind	Confirmed	Not applicable	m-Create
M-CREATE rsp/conf	Not applicable	Not applicable	m-Create
M-DELETE req/ind	Confirmed	Not applicable	m-Delete
M-DELETE rsp/conf	Not applicable	Absent	m-Delete
M-DELETE rsp/conf	Not applicable	Present	m-Linked-Reply

Note – The mapping from the OPERATION and ERROR macros to ROSE is as defined in Recommendation X.219 [8]

7.3 *ACSE user data*

The ACSE protocol (Recommendation X.227 [10]) is described using ASN.1. The "user information" is defined using the EXTERNAL data type.

7.3.1 *A-ASSOCIATE user data*

The encoding of the CMIP user information to be passed to A-ASSOCIATE in the "user information" parameter is defined as follows:

```
CMIP-A-ASSOCIATE-Information {joint-iso-ccitt ms(9) cmip(1) modules(0) aAssociateUserInfo(1)}

DEFINITIONS        ::= BEGIN

FunctionalUnits    ::= BIT STRING {
        multipleObjectSelection          (0),
        filter                           (1),
        multipleReply                    (2),
        extendedService                  (3),
        cancelGet                        (4)
        }

-- Functional unit i is supported if and only if bit i is one

-- Information carried in user-information parameter of A-ASSOCIATE

CMIPUserInfo       ::= SEQUENCE {
        protocolVersion                  [0] IMPLICIT ProtocolVersion DEFAULT { version1 },
        functionalUnits                  [1] IMPLICIT FunctionalUnits DEFAULT {},
        accessControl                    [2] EXTERNAL OPTIONAL,
        userInfo                         [3] EXTERNAL OPTIONAL
        }

ProtocolVersion    ::= BIT STRING {
        version1                         (0),
        version2                         (1)
        }
```

Note – This Recommendation specifies protocol version 2 for technical compatibility with ISO/IEC 9596-1: 1991.

```
END
```

The encoding of other "user information" supplied by the CMISE-service user is not defined by this Recommendation.

7.3.2 *A-ABORT user data*

The encoding of the CMIP user information to be passed to A-ABORT in the "user information" parameter is defined as follows

```
CMIP-A-ABORT-Information {joint-iso-ccitt-ms(9) cmip(1) modules(0) aAbortUserInfo(2)}

DEFINITIONS        ::= BEGIN

-- Information carried in user-information parameter of A-ABORT

CMIPAbortInfo      ::= SEQUENCE {
        abortSource                      [0] IMPLICIT CMIPAbortSource,
        userInfo                         [1] EXTERNAL OPTIONAL
        }
```

```
CMIPAbortSource   ::= ENUMERATED {
        cmiseServiceUser                (0),
        cmiseServiceProvider            (1)
        }

END
```

The encoding of other "user information" supplied by the CMISE-service user is not defined by this Recommendation.

7.4 *CMIP data units*

The protocol is described in terms of Common Management Information Protocol Data Units exchanged between the peer CMISEs. The PDUs are specified using ASN.1 and the Remote Operations Protocol OPERATION and ERROR external macros defined in Recommendation X.219 [8].

```
-- Common Management Information Protocol (CMIP)

    CMIP-1 {joint-iso-ccitt-ms(9) cmip(1) modules(0) protocol(3)}

    DEFINITIONS         ::= BEGIN

-- Remote Operations definitions

    IMPORTS OPERATION, ERROR FROM Remote-Operation-Notation {joint-iso-ccitt remoteOperations(4)
notation(0)}

-- Remote Operations Service definitions

    InvokeIDType FROM Remote-Operations-APDUs {joint-iso-ccitt remoteOperations(4) apdus(1)}

-- Directory Service definitions

    DistinguishedName, RDNSequence FROM InformationFramework {joint-iso-ccitt ds(5) modules(1)
informationFramework(1)};

-- CMISE operations

-- in the following operations, the argument type is mandatory in the corresponding ROSE APDU

-- Action operations (M-ACTION)

    m-Action OPERATION
        ARGUMENT ActionArgument
        ::= localValue 6

    m-Action-Confirmed OPERATION
        ARGUMENT        ActionArgument
        RESULT          ActionResult

-- this result is conditional; for conditions see Recommendation X.710 § 8.3.3.2.9

    ERRORS {
        accessDenied, classInstanceConflict, complexityLimitation, invalidScope, invalidArgumentValue,
        invalidFilter, noSuchAction, noSuchArgument, noSuchObjectClass, noSuchObjectInstance,
        processingFailure, syncNotSupported
        }

    LINKED {    m-Linked-Reply }
        ::= localValue 7

    m-CancelGet OPERATION
        ARGUMENT
        getInvokeId         InvokeIDType
```

```
        RESULT
                ERRORS {    mistypedOperation, noSuchInvokeId, processingFailure }
                ::= localValue 10

-- Create operation (M-CREATE)

        m-Create OPERATION
                ARGUMENT        CreateArgument
                RESULT          CreateResult

-- this result is conditional; for conditions see Recommendation X.710 § 8.3.4.1.3

        ERRORS {
                accessDenied, classInstanceConflict, duplicateManagedObjectInstance, invalidAttributeValue,
                invalidObjectInstance, missingAttributeValue, noSuchAttribute, noSuchObjectClass,
                noSuchObjectInstance, noSuchReferenceObject, processingFailure
                }
                ::= localValue 8

-- Delete operation (M-DELETE)

        m-Delete OPERATION
                Argument        DeleteArgument
                RESULT          DeleteResult

-- this result is conditional; for conditions see Recommendation X.710 § 8.3.5.2.8

        ERRORS {
                accessDenied, classInstanceConflict, complexityLimitation, invalidFilter,
                InvalidScope, noSuchObjectClass, noSuchObjectInstance, processingFailure, syncNotSupported
                }

        LINKED {    m-Linked-Reply }
                ::= localValue 9

-- Event Reporting operations (M-EVENT-REPORT)

        m-EventReport OPERATION
                ARGUMENT        EventReportArgument
                ::= localValue 0

        m-EventReport-Confirmed OPERATION
                ARGUMENT        EventReportArgument
                RESULT          EventReportResult -- optional

        ERRORS {
                invalidArgumentValue, noSuchArgument, noSuchEventType, noSuchObjectClass,
                noSuchObjectInstance, processingFailure
                }
                ::= localValue 1

-- Get operation (M-GET)

        m-Get OPERATION
                ARGUMENT        GetArgument
                RESULT          GetResult

-- this result is conditional; for conditions see Recommendation X.710  § 8.3.1.2.8

        ERRORS {
                accessDenied, classInstanceConflict, complexityLimitation, getListError, invalidFilter,
                invalidScope, no SuchObjectClass, noSuchObjectInstance, operationCancelled,
                processingFailure, syncNotSupported
                }
```

```
        LINKED {      m-Linked-Reply }
              ::= localValue 3

--  Linked operation to M-GET, M-SET (Confirmed), M-ACTION (Confirmed), and M-DELETE

        m-Linked-Reply OPERATION
              ARGUMENT        LinkedReplyArgument
              ::= localValue 2

--  Set operations (M-SET)

        m-Set OPERATION
              ARGUMENT        SetArgument
              ::= localValue 4

        m-Set-Confirmed OPERATION
              ARGUMENT        SetArgument
              RESULT          SetResult

--  this result is conditional; for conditions see Recommendation X.710 § 8.3.2.2.9

        ERRORS {
              accessDenied, classInstanceConflict, complexityLimitation, invalidFilter, invalidScope,
              noSuchObjectClass, noSuchObjectInstance, processingFailure, setListError, syncNotSupported
              }

        LINKED {      m-Linked-Reply }
              ::= localValue 5

--  CMIS error definitions

--  in the following errors, unless otherwise indicated, the parameter type is mandatory in the corresponding
    ROSE APDU

        accessDenied ERROR
              ::= localValue 2

        classInstanceConflict ERROR
              PARAMETER       BaseManagedObjectId
              ::= localValue 19

        complexityLimitation ERROR
              PARAMETER       ComplexityLimitation -- optional
              ::= localValue 20

        duplicateManagedObjectInstance ERROR
              PARAMETER       ObjectInstance
              ::= localValue 11

        getListError ERROR
              PARAMETER       GetListError
              ::= localValue 7

        invalidArgumentValue ERROR
              PARAMETER       InvalidArgumentValue
              ::= localValue 15

        invalidAttributeValue ERROR
              PARAMETER       Attribute
              ::= localValue 6

        invalidFilter ERROR
              PARAMETER       CMISFilter
              ::= localValue 4
```

invalidObjectInstance ERROR
PARAMETER ObjectInstance
::= localValue 17

invalidScope ERROR
PARAMETER Scope
::= localValue 16

missingAttributeValue ERROR
PARAMETER SET OF AttributeId
::= localValue 18

mistypedOperation ERROR
::= localValue 21

noSuchAction ERROR
PARAMETER NoSuchAction
::= localValue 9

noSuchArgument ERROR
PARAMETER NoSuchArgument
::= localValue 14

noSuchAttribute ERROR
PARAMETER AttributeId
::= localValue 5

noSuchEventType ERROR
PARAMETER NoSuchEventType
::= localValue 13

noSuchInvokeId ERROR
PARAMETER InvokeIDType
::= localValue 22

noSuchObjectClass ERROR
PARAMETER ObjectClass
::= localValue 0

noSuchObjectInstance ERROR
PARAMETER ObjectInstance
::= localValue 1

noSuchReferenceObject ERROR
PARAMETER ObjectInstance
::= localValue 12

operationCancelled ERROR
::= localValue 23

processingFailure ERROR
PARAMETER ProcessingFailure -- optional
::= localValue 10

setListError ERROR
PARAMETER SetListError
::= localValue 8

syncNotSupported ERROR
PARAMETER CMISSync
::= localValue 3

-- *Supporting type definitions*

```
AccessControl       ::= EXTERNAL

ActionArgument                          ::= SEQUENCE {
        COMPONENTS OF                   BaseManagedObjectId,
        accessControl                   [5] AccessControl OPTIONAL,
        synchronization                 [6] IMPLICIT CMISSync DEFAULT bestEffort,
        scope                           [7] Scope DEFAULT baseObject,
        filter                          CMISFilter DEFAULT and {},
        actionInfo                      [12] IMPLICIT ActionInfo
        }

ActionError                             ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL,
        actionErrorInfo                 [6] ActionErrorInfo
        }

ActionErrorInfo                         ::= SEQUENCE {

errorStatus                             ENUMERATED {
        accessDenied                    (2),
        noSuchAction                    (9),
        noSuchArgument                  (14),
        invalidArgumentValue            (15)
        },
errorInfo                               CHOICE {
        actionType                      ActionTypeId,
        actionArgument                  [0] NoSuchArgument,
        argumentValue                   [1] InvalidArgumentValue
        } }

ActionInfo                              ::= SEQUENCE {
        actionType                      ActionTypeId,
        actionInfoArg                   [4] ANY DEFINED BY actionType OPTIONAL
        }

ActionReply                              ::= SEQUENCE {
        actionType                      ActionTypeId,
        actionReplyInfo                 [4] ANY DEFINED BY actionType
        }

ActionResult                            ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL,
        actionReply                     [6] IMPLICIT ActionReply OPTIONAL
        }

ActionTypeId                            ::= CHOICE {
        globalForm                      [2] IMPLICIT OBJECT IDENTIFIER,
        localForm                       [3] IMPLICIT INTEGER
        }
```

-- *This Recommendation does not allocate any values for localForm. Where this alternative is used, the permissible values for the integers and their meanings shall be defined as part of the application context in which they are used*

```
Attribute       ::= SEQUENCE {
        attributeId             AttributeId,
        attributeValue          ANY DEFINED BY  attributeId
        }
```

```
AttributeError                  ::= SEQUENCE {

errorStatus                     ENUMERATED {
        accessDenied             (2),
        noSuchAttribute          (5),
        invalidAttributeValue    (6),
        invalidOperation        (24),
        invalidOperator         (25)
        },

modifyOperator                   [2] IMPLICIT ModifyOperator OPTIONAL,
                                 -- present for invalidOperator & invalidOperation
        attributeId              AttributeId,
        attributeValue           ANY DEFINED BY attributeId OPTIONAL
                                 -- absent for setToDefault

        }

AttributeId                     ::= CHOICE {
        globalForm              [0] IMPLICIT OBJECT IDENTIFIER,
        localForm               [1] IMPLICIT INTEGER
        }
```

-- *This Recommendation does not allocate any values for localForm. Where this alternative is used, the permissible values for the integers and their meanings shall be defined as part of the application context in which they are used*

```
AttributeIdError                ::= SEQUENCE {

errorStatus                     ENUMERATED {
        accessDenied            (2),
        noSuchAttribute         (5)             },
        attributeId             AttributeId
        }

BaseManagedObjectId              ::= SEQUENCE {
        baseManagedObjectClass  ObjectClass,
        baseManagedObjectInstance ObjectInstance
        }

CMISFilter                      ::= CHOICE {
        item                    [8] FilterItem,
        and                     [9] IMPLICIT SET OF CMISFilter,
        or                      [10] IMPLICIT SET OF CMISFilter,
        not                     [11] CMISFilter
        }

CMISSync                        ::= ENUMERATED {
        bestEffort              (0),
        atomic                  (1)
        }

ComplexityLimitation            ::= SET {
        scope                   [0] Scope OPTIONAL,
        filter                  [1] CMISFilter OPTIONAL,
        sync                    [2] CMISSync OPTIONAL
        }

CreateArgument                  ::= SEQUENCE {
        managedObjectClass      ObjectClass,
```

```
CHOICE {
        managedObjectInstance           ObjectInstance,
        superiorObjectInstance          [8] ObjectInstance } OPTIONAL,
        accessControl                   [5] AccessControl OPTIONAL,
        referenceObjectInstance         [6] ObjectInstance OPTIONAL,
        attributeList                   [7] IMPLICIT SET OF Attribute OPTIONAL
        }
CreateResult                            ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
                                        -- shall be returned if omitted from CreateArgument
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL,
        attributeList                   [6] IMPLICIT SET OF Attribute OPTIONAL
        }

DeleteArgument                          ::= SEQUENCE {
        COMPONENTS OF                   BaseManagedObjectId,
        accessControl                   [5] AccessControl OPTIONAL,
        synchronization                 [6] IMPLICIT CMISSync DEFAULT bestEffort,
        scope                           [7] Scope DEFAULT baseObject,
        filter                          CMISFilter DEFAULT and {}
        }

DeleteError                             ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL,
        deleteErrorInfo                 [6] ENUMERATED { accessDenied (2)
        } }

DeleteResult                            ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL
        }

EventReply                              ::= SEQUENCE {
        eventType                       EventTypeId,
        eventReplyInfo                  [8] ANY DEFINED BY eventType OPTIONAL
        }

EventReportArgument                     ::= SEQUENCE {
        managedObjectClass              ObjectClass,
        managedObjectInstance           ObjectInstance,
        eventTime                       [5] IMPLICIT GeneralizedTime OPTIONAL,
        eventType                       EventTypeId,
        eventInfo                       [8] ANY DEFINED BY eventType OPTIONAL
        }

EventReportResult                       ::= SEQUENCE {
        managedObjectClass              ObjectClass OPTIONAL,
        managedObjectInstance           ObjectInstance OPTIONAL,
        currentTime                     [5] IMPLICIT GeneralizedTime OPTIONAL,
        eventReply                      EventReply OPTIONAL
        }

EventTypeId                             ::= CHOICE {
        globalForm                      [6] IMPLICIT OBJECT IDENTIFIER,
        localForm                       [7] IMPLICIT INTEGER
        }
```

-- *This Recommendation does not allocate any values for localForm. Where this alternative is used, the permissible values for the integers and their meanings shall be defined as part of the application context in which they are used*

```
FilterItem                  ::= CHOICE {
    equality                [0] IMPLICIT Attribute,
    substrings              [1] IMPLICIT SEQUENCE OF CHOICE {
    initialString           [0] IMPLICIT SEQUENCE {
                                attributeId     AttributeId,
                                string          ANY DEFINED BY attributeId },
    anyString               [1] IMPLICIT SEQUENCE {
                                attributeId     AttributeId,
                                string          ANY DEFINED BY attributeId },
    finalString             [2] IMPLICIT SEQUENCE {
                                attributeId     AttributeId,
                                string          ANY DEFINED BY attributeId} },
    greaterOrEqual          [2] IMPLICIT Attribute,   -- asserted value ≥ attribute value
    lessOrEqual             [3] IMPLICIT Attribute,   -- asserted value ≥ attribute value
    present                 [4] AttributeId,
    subsetOf                [5] IMPLICIT Attribute,   -- asserted value is a subset of
                                                      attribute value
    supersetOf              [6]IMPLICIT Attribute,    -- asserted value is a superset of
                                                      attribute value
    nonNullSetIntersection  [7] IMPLICIT Attribute
    }

GetArgument                 ::= SEQUENCE {
COMPONENTS OF               BaseManagedObjectId,
accessControl               [5] AccessControl OPTIONAL,
synchronization             [6] IMPLICIT CMISSync DEFAULT bestEffort,
scope                       [7] Scope DEFAULT baseObject,
filter                      CMISFilter DEFAULT and {},
attributeIdList             [12] IMPLICIT SET OF AttributeId OPTIONAL
}

GetInfoStatus               ::= CHOICE {
attributeIdError            [0] IMPLICIT AttributeIdError,
attribute                   [1] IMPLICIT Attribute
}

GetListError                ::= SEQUENCE {
managedObjectClass          ObjectClass OPTIONAL,
managedObjectInstance       ObjectInstance OPTIONAL,
currentTime                 [5] IMPLICIT GeneralizedTime OPTIONAL,
getInfoList                 [6] IMPLICIT SET OF GetInfoStatus
}

GetResult                   ::= SEQUENCE {
managedObjectClass          ObjectClass OPTIONAL,
managedObjectInstance       ObjectInstance OPTIONAL,
currentTime                 [5] IMPLICIT GeneralizedTime OPTIONAL,
attributeList               [6] IMPLICIT SET OF Attribute OPTIONAL
}
```

```
        InvalidArgumentValue            ::= CHOICE {
        actionValue                     [0] IMPLICIT ActionInfo,
        eventValue                      [1] IMPLICIT SEQUENCE {
                                            eventType  EventTypeId,
                                            eventInfo  [8] ANY DEFINED BY eventType OPTIONAL }
        }

    LinkedReplyArgument                 ::= CHOICE {
        getResult                       [0] IMPLICIT GetResult,
        getListError                    [1] IMPLICIT GetListError,
        setResult                       [2] IMPLICIT SetResult,
        setListError                    [3] IMPLICIT SetListError,
        actionResult                    [4] IMPLICIT ActionResult,
        processingFailure               [5] IMPLICIT ProcessingFailure,
        deleteResult                    [6] IMPLICIT DeleteResult,
        actionError                     [7] IMPLICIT ActionError,
        deleteError                     [8] IMPLICIT DeleteError
        }

    ModifyOperator                      ::= INTEGER {
        replace                         (0),
        addValues                       (1),
        removeValues                    (2),
        setToDefault                    (3)
        }

    NoSuchAction                        ::= SEQUENCE {
        managedObjectClass              ObjectClass,
        actionType                      ActionTypeId
        }

    NoSuchArgument                      ::= CHOICE {
        actionId                        [0] IMPLICIT SEQUENCE {
                                            managedObjectClass  ObjectClass OPTIONAL,
                                            actionType          ActionTypeId },
        eventId                         [1] IMPLICIT SEQUENCE {
                                            managedObjectClass  ObjectClass OPTIONAL,
                                            eventType           EventTypeId }
        }

    NoSuchEventType                     ::= SEQUENCE {
        managedObjectClass              ObjectClass,
        eventType                       EventTypeId
        }

    ObjectClass                         ::= CHOICE {
        globalForm                      [0] IMPLICIT OBJECT IDENTIFIER,
        localForm                       [1] IMPLICIT INTEGER
        }
```

-- *This Recommendation does not allocate any values for localForm. Where this alternative is used, the permissible values for the integers and their meanings shall be defined as part of the application context in which they are used*

```
    ObjectInstance                      ::= CHOICE {
        distinguishedName               [2] IMPLICIT DistinguishedName,
        nonSpecificForm                 [3] IMPLICIT OCTET STRING,
        localDistinguishedName          [4] IMPLICIT RDNSequence
        }
```

```
-- localDistinguishedName is that portion of the distinguished name that is necessary to unambiguously
   identify the managed object within the context of communication between the open systems

   ProcessingFailure                ::= SEQUENCE {
         managedObjectClass         ObjectClass,
         managedObjectInstance      ObjectInstance OPTIONAL,
         specificErrorInfo          [5] SpecificErrorInfo
         }

   Scope                            ::= CHOICE {INTEGER {
         baseObject                 (0),
         firstLevelOnly             (1),
         wholeSubtree               (2) },
         individualLevels           [1] IMPLICIT INTEGER,    -- POSITIVE integer indicates the
                                                             level to be selected
         baseToNthLevel             [2] IMPLICIT INTEGER }   -- POSITIVE integer N indicates
                                                             that the range of levels
                                                             -- (0 - N) is to be selected

-- with individualLevels and baseToNthLevel, a value of 0 has the same semantics as baseObject
-- with individualLevels, a value of 1 has the same semantics as firstLevelOnly

   SetArgument                      ::= SEQUENCE {
         COMPONENTS OF              BaseManagedObjectId,
         accessControl              [5] AccessControl OPTIONAL,
         synchronization            [6] IMPLICIT CMISSync DEFAULT bestEffort,
         scope                      [7] Scope DEFAULT baseObject,
         filter                     CMISFilter DEFAULT and { },
         modificationList           [12] IMPLICIT SET OF SEQUENCE {
         modifyOperator             [2] IMPLICIT ModifyOperator DEFAULT replace,
         attributeId                AttributeId,
         attributeValue             ANY DEFINED BY attributeId OPTIONAL -- absent for
                                    setToDefault
         } }

   SetInfoStatus                    ::= CHOICE {
         attributeError             [0] IMPLICIT AttributeError,
         attribute                  [1] IMPLICIT Attribute
         }

   ListError                        ::= SEQUENCE {
         managedObjectClass         ObjectClass OPTIONAL,
         managedObjectInstance      ObjectInstance OPTIONAL,
         currentTime                [5] IMPLICIT GeneralizedTime OPTIONAL,
         setInfoList                [6] IMPLICIT SET OF SetInfoStatus
         }

   SetResult                        ::= SEQUENCE {
         managedObjectClass         ObjectClass OPTIONAL,
         managedObjectInstance      ObjectInstance OPTIONAL,
         currentTime                [5] IMPLICIT GeneralizedTime OPTIONAL,
         attributeList              [6] IMPLICIT SET OF Attribute OPTIONAL
         }

   SpecificErrorInfo                ::= SEQUENCE {
         errorId                    OBJECT IDENTIFIER,
         errorInfo                  ANY DEFINED BY errorId
         }
   END -- End of CMIP syntax definitions
```

7.5 *Definition of abstract syntax for CMIP*

This Recommendation assigns the ASN.1 object identifier value

{joint-iso-ccitt ms(9) cmip(1) cmip-pci(1) abstractSyntax(4)}

as an abstract syntax name for the set of presentation data values, each of which is either a value of the ASN.1 type

Remote-Operations-APDUs.ROSEapdus

as defined in Recommendation X.229 [11] with the argument component filled according to the definitions in CMIP-1, or a value of one of the ASN.1 types:

- CMIP-A-ASSOCIATE-Information.CMIPUserInfo;

- CMIP-A-ABORT-Information.CMIPAbortInfo.

The corresponding ASN.1 object descriptor value shall be

"CMIP-PCI".

This abstract syntax is defined to include all data types resolved by the ANY DEFINED BY X productions, in which X is of type OBJECT IDENTIFIER.

The ASN.1 object identifier and object descriptor values

{joint-iso-ccitt asn1(1) basic-encoding(1)} and "Basic Encoding of single ASN.1 type"

(assigned to an object in Recommendation X.209 [4]) can be used as a transfer syntax name with this abstract syntax.

7.5.1 *Extensibility rules*

7.5.1.1 When processing incoming CMIP-A-ASSOCIATE-Information, the accepting CMIPM shall

- ignore all tagged values that are not defined in the abstract syntax of this Recommendation; and
- ignore all unknown bit name assignments within a BIT STRING.

7.5.1.2 The abstract syntax name may be used when the presentation data values are modified to include:

- new system management operations;
- new tagged elements within a SET or SEQUENCE;
- new bit name assigments within a BIT STRING;
- new named numbers for an INTEGER; and
- new named enumerations within an ENUMERATED.

8 Conformance

A system claiming to implement the procedures specified in this standard shall comply with the requirements in §§ 8.1 and 8.2.

8.1 *Static requirements*

The system shall

a) support the kernel functional unit defined in Recommendation X.710 [12], and the facilities implied by that functional unit;

b) support the transfer syntax derived from the encoding rules specified in Recommendation X.209 [4] and named

{joint-iso-ccitt asn1(1) basic-encoding(1)}

for the purpose of generating and interpreting CMISE protocol information as defined by the abstract syntax

"CMIP-PCI"

for the functional units supported:

c) support the ACSE protocol defined in Recommendation X.227 [10], to establish and to release an association;

d) support the rules specified in annex A in any application context that includes CMISE as one of the ASEs;

e) support association class 3 of the ROSE protocol defined in Recommendation X.229 [11];

f) support the multiple reply functional unit if the multiple object selction functional unit is supported.

8.2 *Dynamic requirements*

The system shall

a) follow the procedures relevant to each functional unit that the system claims to implement;

b) when used, verify the optional security parameters defined in the CMIP PDUs;

c) when the extended service functional unit is supported, support the presentation protocol defined in Recommendation X.226 [9], as required by the application context;

d) when scoping is provided, support the multiple reply functional unit.

ANNEX A

(This annex does not form an integral part of this Recommendation)

Association rules for CMISE

A.1 *ACSE, session and presentation requirements*

A.1.1 CMISE requires the kernel presentation functional unit as defined in Recommendation X.216 [6].

A.1.2 CMISE requires the kernel and full duplex session functional units as defined in Recommendation X.215 [5].

A.1.3 CMISE requires the normal mode of ACSE and presentation services as defined in Recommendations X.227 [10] and X.216 [6].

A.2 *Association initialisation rules*

A.2.1 *Request*

The CMISE-service-user that initiates the association establishment shall provide the A-ASSOCIATE user information defined by Recommendation X.710 [12]. The CMIP user information shall be made available to the CMIPM which shall

a) construct CMIPUserInfo from the information supplied;

b) set the protocol version parameter within CMIPUserInfo by setting the bit corresponding to each version supported;

c) include CMIPUserInfo as a separate EXTERNAL in the user information parameter of the A-ASSOCIATE request primitive;

d) wait for the user information specific to CMIS to be returned in the A-ASSOCIATE confirm primitive.

A.2.2 *Indication*

On receipt of an A-ASSOCIATE indication primitive, the CMIPUserInfo parameter shall be made available to the CMIPM which shall

a) check that at least one of the proposed protocol version can be supported;

b) verify that the optional access control parameter is valid;

c) if any of the checks fail, the association shall be rejected by setting the reason for failure parameter in the A-ASSOCIATE response primitive to "rejected by responder (permanent)". The association is not established and that instance of the CMIPM shall cease to exist;

d) if the above checks succeed, the following information, if present in CMIPUserInfo, shall be made available to the CMISE-service-user: functional units supported by the CMISE-service-provider, access control and user information. The CMIPM shall wait for the response from the CMISE-service-user.

A.2.3 *Response*

The A-ASSOCIATE response primitive indicating "accepted" or "rejected", and which if accepted, includes the functional units, access control and user information parameters, shall be made available to the CMIPM which shall

a) construct CMIPUserInfo required for the response. The CMIPUserInfo shall include the version parameter indicating all versions of CMIP that are supported;

b) include CMIPUserInfo as a separate EXTERNAL in the user information parameter of the A-ASSOCIATE response primitive;

c) if the association response indicates "accepted", the protocol version agreed to is the version corresponding to the highest number supported by both CMIPMs. The CMIPM shall then be ready to accept CMISE indication primitives;

d) if the association response indicates "rejected", that instance of the CMIPM shall cease to exist.

A.2.4 *Confirmation*

On receipt of the A-ASSOCIATE confirmation primitive, the CMIPUserInfo parameter shall be made available to the CMIPM which shall

a) if the association confirmation indicates success, the association is established and the functional units, access control and user information parameters, if present in the confirmation, are made available to the association-initiator. The functional units agreed to correspond to those for which both CMISE-service-users indicated support and the protocol version is the highest version number supported by both CMIPMs;

b) if the association confirmation indicates failure, the association is not established and that instance of the CMIPM shall cease to exist.

A.3 *Association release rules*

Either CMISE-service-user may initiate an association release.

A.3.1 *Request*

On receipt of a request for association release, the necessary A-RELEASE parameters shall be made available to the CMIPM which shall cease to accept service requests and wait for the confirmation of the release of the association.

A.3.2 *Indication*

On receipt of an A-RELEASE indication primitive, the necessary A-RELEASE indication parameters shall be made available to the responding CMIPM which shall wait for the association release response.

A.3.3 *Response*

On receipt of an association release response from the responding CMISE-service-user, the necessary A-RELEASE response parameters shall be made available to the responding CMIPM. Thereafter, that instance of the CMIPM shall cease to exist.

A.3.4 *Confirmation*

On receipt of an A-RELEASE confirm primitive, the necessary A-RELEASE confirm parameters shall be made available to the initiating CMIPM. Thereafter, that instance of the CMIPM shall cease to exist.

A.4 *Association abort rules*

Either CMISE-service-user may initiate an abrupt termination of the association.

On the basis of local information, if the ability of the underlying services to convey unlimited user information by A-ABORT does not exist, the CMIPAbortInfo parameter may not be included in the A-ABORT service primitives.

A.4.1 *A-ABORT request*

On receipt of a request to abort the association, the necessary A-ABORT request parameters including the A-ABORT user information defined by Recommendation X.710 [12] shall be made available to the CMIPM which shall

a) construct CMIPAbortInfo from the information supplied;

b) set the abort source parameter within CMIPUserInfo to CMISE-service-user;

c) include CMIPAbortInfo as a separate field in the user information parameter of the A-ABORT request primitive;

d) thereafter, that instance of the CMIPM shall cease to exist.

A.4.2 *A-ABORT Indication*

On receipt of an A-ABORT indication primitive, the necessary A-ABORT indication parameters including CMIPAbortInfo shall be made available to the CMIPM. Thereafter, that instance of the CMIPM shall cease to exist.

A.4.3 *A-P-ABORT Indication*

On receipt of an A-P-ABORT indication primitive, the necessary A-P-ABORT indication parameters shall be made available to the CMIPM. Thereafter, that instance of the CMIPM shall cease to exist.

A.4.4 *CMIP protocol error*

On detecting a protocol error, the CMIPM shall

a) construct CMIPAbortInfo with the abort source parameter set to CMISE-service-provider;

b) indicate to the CMISE-service-user that a protocol error has occurred;

c) include CMIPAbortInfo as a separate field in the user information parameter of the A-ABORT request primitive;

d) thereafter, that instance of the CMIPM shall cease to exist.

ANNEX B

(This annex does not form an integral part of this Recommendation)

Expanded ASN.1 syntax

This annex describes how the OPERATION and ERROR macros of Recommendation X.219 [8] are expanded into ASN.1 data types and subtypes.

If any inconsistencies exist between these definitions and the definitions in § 7, then the definitions in § 7 take precedence.

```
-- Common Management Information Protocol (CMIP)

CMIP-1 {joint-iso-ccitt ms(9) cmip(1) modules(0) protocol(3)}

DEFINITIONS        ::= BEGIN

-- Remote Operations definitions

IMPORTS OPERATION, ERROR FROM Remote-Operation-Notation {joint-iso-ccitt remoteOperations(4)
notation(0)}

-- Directory Service definitions

DistinguishedName, RDNSequence FROM InformationFramework {joint-iso-ccitt ds(5) modules(1)
informationFramework(1)};

--  CMISE operations

ROSEapdus                       ::= CHOICE {
    roiv-apdu                   [1] IMPLICIT ROIVapdu,
    rors-apdu                   [2] IMPLICIT RORSapdu,
    roer-apdu                   [3] IMPLICIT ROERapdu,
    rorj-apdu                   [4] IMPLICIT RORJapdu
    }

ROIVapdu                        ::= SEQUENCE {
    invokeID                    InvokeIDType,
    linked-ID                   [0] IMPLICIT InvokeIDType OPTIONAL,
    operation-value             OPERATION,
    argument                    ANY DEFINED BY operation-value OPTIONAL
    }

RORSapdu                        ::= SEQUENCE {
    invokeID                    InvokeIDType,
    SEQUENCE {                  operation-value     OPERATION,
                                result              ANY DEFINED BY
                                                    operation-value } OPTIONAL
    }
```

```
ROERapdu                        ::= SEQUENCE {
        invokeID                InvokeIDType,
        error-value             ERROR,
        parameter               ANY DEFINED BY      error-value OPTIONAL
        }

RORJapdu                        ::= SEQUENCE {
        invokeID    CHOICE {    InvokeIDType,
                                NULL },
        problem     CHOICE {    [0] IMPLICIT GeneralProblem,
                                [1] IMPLICIT InvokeProblem,
                                [2] IMPLICIT ReturnResultProblem,
                                [3] IMPLICIT ReturnErrorProblem }
        }

InvokeIDType    ::= INTEGER

-- The use of the GeneralProblem, ReturnResultProblem, and ReturnErrorProblem codes are a local issue.

GeneralProblem                  ::= INTEGER {
        unrecognisedAPDU        (0),        -- ROSE-provider detected
        mistypedAPDU            (1),
        badlyStructuredAPDU     (2)
        }

InvokeProblem                   ::= INTEGER {
        duplicateInvocation     (0),        -- ROSE-user detected
        unrecognisedOperation   (1),
        mistypedArgument        (2),
        resourceLimitation      (3),
        initiatorReleasing      (4),
        unrecognisedLinkedID    (5),
        linkedResponseUnexpected (6),
        unexpectedChildOperation (7)
        }

ReturnResultProblem             ::= INTEGER {
        unrecognisedInvocation  (0)         -- ROSE-user detected
        resultResponseUnexpected (1),
        mistypedResult          (2)
        }

ReturnErrorProblem              ::= INTEGER {
        unrecognisedInvocation  (0),        -- ROSE-user detected
        errorResponseUnexpected (1),
        unrecognisedError       (2),
        unexpectedError         (3),
        mistypedParameter       (4)
        }

-- This part of the ASN.1 specification provides a definition of the InvokeProblem subtype used by CMIP.

InvokeProblem-CMIPUser          ::= InvokeProblem (
        duplicateInvocation     |
        unrecognisedOperation   |
        mistypedArgument        |
        resourceLimitation
        )
```

```
-- This part of the ASN.1 specification provides a definition of ROIVapdu and RORSapdu subtypes used by
   CMIP. The subtypes of the ROIVapdu define the allowed values of the operation-value and argument
   defined by that operation-value for all CMIP notifications and operations. The subtypes of the RORSapdu
   define the allowed values of the operation-value and result defined by that operation-value for all CMIP
   notifications and operations.

m-Action OPERATION                  ::= localValue 6

ROIV-m-Action                       ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Action),
    argument                        (INCLUDES ActionArgument) } )

m-Action-Confirmed OPERATION        ::= localValue 7

ROIV-m-Action-Confirmed             ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Action-Confirmed),
    argument                        (INCLUDES ActionArgument) } )

RORS-m-Action-Confirmed             ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invokeID                        PRESENT,
    -- result sequence -- (WITH COMPONENTS
    { operation-value               (m-Action-Confirmed),
    result                          (INCLUDES ActionResult) } )
    -- required only if there is a single reply to the ROIV-m-Action-Confirmed ROIVapdu and data is to be
    returned in the RORSapdu
    } )

m-Cancel-Get OPERATION              ::= localValue 10

ROIV-m-Cancel-Get                   ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Cancel-Get),
    argument                        (INCLUDES InvokeIDType)
    } )

RORS-m-Cancel-Get                   ::= RORSapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    -- There is no result sequence for RORS-m-Cancel-Get
    } )

m-Create OPERATION                  ::= localValue 8

ROIV-m-Create                       ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Create),
    argument                        (INCLUDES CreateArgument) } )

RORS-m-Create                       ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invoke-ID                       PRESENT,
    -- result sequence  -- (WITH COMPONENTS
    { operation-value               (m-Create),
    result                          (INCLUDES CreateResult) } )
    } )
```

```
m-Delete OPERATION                  ::= localValue 9

ROIV-m-Delete                       ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Delete),
    argument                        (INCLUDES DeleteArgument) } )

RORS-m-Delete                       ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invokeID                        PRESENT,
    -- result sequence -- (WITH COMPONENTS
    { operation-value               (m-Delete),
    result                          (INCLUDES DeleteResult) } )
    -- required only if there is a single reply to the ROIV-m-DeleteROIVapdu and data is to be returned in the
    RORSapdu
    } )

m-EventReport OPERATION             ::= localValue 0

ROIV-m-EventReport                  ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-EventReport),
    argument                        (INCLUDES EventReportArgument) } )

m-EventReport-Confirmed OPERATION   ::= localValue 1

ROIV-m-EventReport-Confirmed        ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-EventReport-Confirmed),
    argument                        (INCLUDES EventReportArgument) } )

RORS-m-EventReport-Confirmed        ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invokeID                        PRESENT,
    -- result sequence -- (WITH COMPONENTS
    { operation-value               (m-EventReport-Confirmed),
    result                          (INCLUDES EventReportResult) } )
    -- required only if data is to be returned in the RORSapdu
    } )

m-Get OPERATION                     ::= localValue 3

ROIV-m-Get                          ::= ROIVapdu (WITH COMPONENTS
    { invokeID                      PRESENT,
    linked-ID                       ABSENT,
    operation-value                 (m-Get),
    argument                        (INCLUDES GetArgument) } )

RORS-m-Get                          ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invokeID                        PRESENT,
    -- result sequence -- (WITH COMPONENTS
    { operation-value               (m-Get),
    result                          (INCLUDES GetResult) } )
    -- required only if there is a single reply to the ROIV-m-Get ROIVapdu
    } )
```

```
m-Linked-Reply OPERATION                ::= localValue 2

ROIV-m-Linked-Reply                     ::= ROIVapdu (WITH COMPONENTS
        { invokeID                      PRESENT,
        linked-ID                       PRESENT,
        operation-value                 (m-Linked-Reply),
        argument                        (INCLUDES LinkedReplyArgument) } )
```

-- *This part of the ASN.1 specification provides a definition of ROIV-m-Linked-Reply subtypes used by CMIP. The subtypes of the ROIV-m-Linked-Reply ROIVapdu define the allowed values of the argument defined by the operation-value for the specific CMIP linked reply operations.*

```
ROIV-m-Linked-Reply-Action              ::= ROIV-m-Linked-Reply (WITH COMPONENTS
        { invokeID                      PRESENT,
        linked-ID                       PRESENT,
        operation-value                 (m-Linked-Reply),
        argument                        (INCLUDES LinkedReplyArgument (WITH COMPONENTS
                                        getResult               ABSENT,
                                        getListError            ABSENT,
                                        setResult               ABSENT,
                                        setListError            ABSENT,
                                        actionResult            PRESENT,
                                        processingFailure       PRESENT,
                                        deleteResult            ABSENT,
                                        actionError             PRESENT,
                                        deleteError             ABSENT )
        ) } )

ROIV-m-Linked-Reply-Delete              ::= ROIV-m-Linked-Reply (WITH COMPONENTS
        { invokeID                      PRESENT,
        linked-ID                       PRESENT,
        operation-value                 (m-Linked-Reply),
        argument                        (INCLUDES LinkedReplyArgument (WITH COMPONENTS
                                        getResult               ABSENT,
                                        getListError            ABSENT,
                                        setResult               ABSENT,
                                        setListError            ABSENT,
                                        actionResult            ABSENT,
                                        processingFailure       PRESENT,
                                        deleteResult            PRESENT,
                                        actionError             ABSENT,
                                        deleteError             PRESENT )
        ) } )

ROIV-m-Linked-Reply-Get                 ::= ROIV-m-Linked-Reply (WITH COMPONENTS
        { invokeID                      PRESENT,
        linked-ID                       PRESENT,
        operation-value                 (m-Linked-Reply),
        argument                        (INCLUDES LinkedReplyArgument (WITH COMPONENTS
                                        getResult               PRESENT,
                                        getListError            PRESENT,
                                        setResult               ABSENT,
                                        setListError            ABSENT,
                                        actionResult            ABSENT,
                                        processingFailure       PRESENT,
                                        deleteResult            ABSENT,
                                        actionError             ABSENT,
                                        deleteError             ABSENT )
        ) } )
```

```
ROIV-m-Linked-Reply-Set          ::= ROIV-m-Linked-Reply (WITH COMPONENTS
    { invokeID                   PRESENT,
    linked-ID                    PRESENT,
    operation-value              (m-Linked-Reply),
    argument                     (INCLUDES LinkedReplyArgument (WITH COMPONENTS
                                 getResult          ABSENT,
                                 getListError       ABSENT,
                                 setResult          PRESENT
                                 setListError       PRESENT,
                                 actionResult       ABSENT,
                                 processingFailure  PRESENT,
                                 deleteResult       ABSENT,
                                 actionError        ABSENT,
                                 deleteError        ABSENT )
    ) } )

m-Set OPERATION                  ::= localValue 4

ROIV-m-Set                       ::= ROIVapdu (WITH COMPONENTS
    { invokeID                   PRESENT,
    linked-ID                    ABSENT,
    operation-value              (m-Set),
    argument                     (INCLUDES SetArgument) } )

m-Set-Confirmed OPERATION        ::= localValue 5

ROIV-m-Set-Confirmed             ::= ROIVapdu (WITH COMPONENTS
    { invokeID                   PRESENT,
    linked-ID                    ABSENT,
    operation-value              (m-Set-Confirmed),
    argument                     (INCLUDES SetArgument) } )

RORS-m-Set-Confirmed             ::= RORSapdu (WITH COMPONENTS
    { ... ,
    invokeID                     PRESENT,
    -- result sequence -- (WITH COMPONENTS
    { operation-value            (m-Set-Confirmed),
    result                       (INCLUDES SetResult) } )
    -- required only if there is a single reply to the ROIV-m-Set-Confirmed ROIVapdu and data is to be returned in the RORSapdu
    } )
```

-- *This part of the ASN.1 specification provides a definition of ROERapdu subtypes used by CMIP. The subtypes of the ROERapdu define the allowed values of the error value and parameter defined by that error-value for all CMIP notifications and operations.*

```
accessDenied ERROR               ::= localValue 2

ROER-accessDenied                ::= ROERapdu (WITH COMPONENTS
    { invokeID                   PRESENT,
    error-value                  (accessDenied) } )
```

-- *This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed, ROIV-m-Action-Confirmed, ROIV-m-Create and ROIV-m-Delete ROIVapdus*

```
classInstanceConflict ERROR      ::= localValue 19

ROER-classInstanceConflict       ::= ROERapdu (WITH COMPONENTS
    { invokeID                   PRESENT,
    error-value                  (classInstanceConflict),
    parameter                    (INCLUDES BaseManagedObjectId) } )
```

```
-- This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed,
   ROIV-m-Action-Confirmed, ROIV-m-Create and ROIV-m-Delete ROIVapdus

   complexityLimitation ERROR          ::= localValue 20

   ROER-complexityLimitation           ::= ROERapdu (WITH COMPONENTS
         { invokeID                    PRESENT,
         error-value                   (complexityLimitation),
         parameter                     (INCLUDES ComplexityLimitation) OPTIONAL } )

-- This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed,
   ROIV-m-Action-Confirmed and ROIV-m-Delete ROIVapdus

   duplicateManagedObjectInstance ERROR    ::= localValue 11

   ROER-duplicateManagedObjectInstance     ::= ROERapdu (WITH COMPONENTS
         { invokeID                        PRESENT,
         error-value                       (duplicateManagedObjectInstance),
         parameter                         (INCLUDES ObjectInstance) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

   getListError ERROR                  ::= localValue 7

   ROER-getListError                   ::= ROERapdu (WITH COMPONENTS
         { invokeID                    PRESENT,
         error-value                   (getListError),
         parameter                     (INCLUDES GetListError) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Get ROIVapdu

   invalidArgumentValue ERROR          ::= localValue 15

   ROER-invalidArgumentValue           ::= ROERapdu (WITH COMPONENTS
         { invokeID                    PRESENT,
         error-value                   (invalidArgumentValue),
         parameter                     (INCLUDES InvalidArgumentValue) } )

-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed
   and ROIV-m-Action-Confirmed ROIVapdus

   invalidAttributeValue ERROR         ::= localValue 6

   ROER-invalidAttributeValue          ::= ROERapdu (WITH COMPONENTS
         { invokeID                    PRESENT,
         error-value                   (invalidAttributeValue),
         parameter                     (INCLUDES Attribute) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

   invalidFilter ERROR                 ::= localValue 4

   ROER-invalidFilter                  ::= ROERapdu (WITH COMPONENTS
         { invokeID                    PRESENT,
         error-value                   (invalidFilter),
         parameter                     (INCLUDES CMISFilter) } )
```

```
-- This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed,
-- ROIV-m-Action-Confirmed and ROIV-m-Delete ROIVapdus

   invalidObjectInstance ERROR          ::= localValue 17

   ROER-invalidObjectInstance           ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (invalidObjectInstance),
         parameter                      (INCLUDES ObjectInstance) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

   invalidScope ERROR                   ::= localValue 16

   ROER-invalidScope                    ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (invalidScope),
         parameter                      (INCLUDES Scope) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed,
-- ROIV-m-Action-Confirmed and ROIV-m-Delete ROIVapdus

   missingAttributeValue ERROR          ::= localValue 18

   ROER-missingAttributeValue           ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (missingAttributeValue),
         parameter                      (INCLUDES SET OF AttributeId) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

   mistypedOperation ERROR              ::= localValue 21

   ROER-mistypedOperation               ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (mistypedOperation) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Cancel-Get ROIVapdu

   noSuchAction ERROR                   ::= localValue 9

   ROER-noSuchAction                    ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (noSuchAction),
         parameter                      (INCLUDES NoSuchAction) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Action-Confirmed ROIVapdu

   noSuchArgument ERROR                 ::= localValue 14

   ROER-noSuchArgument                  ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (noSuchArgument),
         parameter                      (INCLUDES NoSuchArgument) } )

-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed and
-- ROIV-m-Action-Confirmed ROIVapdus

   noSuchAttribute ERROR                ::= localValue 5

         ROER-noSuchAttribute           ::= ROERapdu (WITH COMPONENTS
         { invokeID                     PRESENT,
         error-value                    (noSuchAttribute),
         parameter                      (INCLUDES AttributeId) } )
```

```
-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

noSuchEventType ERROR                ::= localValue 13

ROER-noSuchEventType                 ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (noSuchEventType),
        parameter                    (INCLUDES NoSuchEventType) } )

-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed ROIVapdu

noSuchInvokeId ERROR                 ::= localValue 22

ROER-noSuchInvokeId                  ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (noSuchInvokeId),
        parameter                    (INCLUDES InvokeIdType) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Cancel-Get ROIVapdu

noSuchObjectClass ERROR              ::= localValue 0

ROER-noSuchObjectClass               ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (noSuchObjectClass ),
        parameter                    (INCLUDES ObjectClass) } )

-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed, ROIV-m-Get,
-- ROIV-m-Set-Confirmed, ROIV-m-Action-Confirmed, ROIV-m-Create, and ROIV-m-Delete ROIVapdus

noSuchObjectInstance ERROR           ::= localValue 1

ROER-noSuchObjectInstance            ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (noSuchObjectInstance),
        parameter                    (INCLUDES ObjectInstance) } )

-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed, ROIV-m-Get,
-- ROIV-m-Set-Confirmed, ROIV-m-Action-Confirmed, ROIV-m-Create, and ROIV-m-Delete ROIVapdus

noSuchReferenceObject ERROR          ::= localValue 12

ROER-noSuchReferenceObject           ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (noSuchReferenceObject),
        parameter                    (INCLUDES ObjectInstance) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Create ROIVapdu

operationCancelled ERROR             ::= localValue 23

ROER-operationCancelled              ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (operationCancelled) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Get ROIVapdu

processingFailure ERROR              ::= localValue 10

ROER-processingFailure               ::= ROERapdu (WITH COMPONENTS
        { invokeID                   PRESENT,
        error-value                  (processingFailure),
        parameter                    (INCLUDES ProcessingFailure) OPTIONAL } )
```

```
-- This ROERapdu may only be returned in response to the ROIV-m-EventReport-Confirmed, ROIV-m-Get,
   ROIV-m-Set-Confirmed, ROIV-m-Action-Confirmed, ROIV-m-Create, and ROIV-m-Delete ROIVapdus

   setListError ERROR                ::= localValue 8

   ROER-setListError                 ::= ROERapdu (WITH COMPONENTS
         { invokeID                  PRESENT,
         error-value                 (setListError),
         parameter                   (INCLUDES SetListError) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Set-Confirmed ROIVapdu

   syncNotSupported ERROR            ::= localValue 3

   ROER-syncNotSupported             ::= ROERapdu (WITH COMPONENTS
         { invokeID                  PRESENT,
         error-value                 (syncNotSupported),
         parameter                   (INCLUDES CMISSync) } )

-- This ROERapdu may only be returned in response to the ROIV-m-Get, ROIV-m-Set-Confirmed,
   ROIV-m-Action-Confirmed and ROIV-m-Delete ROIVapdus

-- To complete the abstract syntax specification provided in this annex, the definitions of the supporting
   types in § 7.4 are incorporated by reference

   END -- of CMIP syntax definitions
```

ANNEX C

(This annex does not form an integral part of this Recommendation)

Examples of CMISE ROSE APDUs

This annex provides some examples of the complete expansion of ROSE APDUs carrying CMIP information.

These examples are provided as guidance for users of this Recommendation.

```
-- ROIVapdu for the CMISE confirmed action operation.

   ROIVapdu-example                  ::= [1] IMPLICIT SEQUENCE {
         invokeID                    InvokeIDType,
         operation-value             INTEGER {m-Action-Confirmed (7)},
         argument                    SEQUENCE {
               COMPONENTS OF         BaseManagedObjectId,
               accessControl         [5] AccessControl OPTIONAL,
               synchronization       [6] IMPLICIT CMISSync OPTIONAL,
               scope                 [7] Scope DEFAULT baseObject,
               filter                CMISFilter DEFAULT and {},
         actionInfo                  [12] IMPLICIT SEQUENCE (
         actionType                  ActionTypeId,
         actionInfoArg               [4] ANY DEFINED BY actionType OPTIONAL
         } }
```

-- RORSapdu for the CMISE confirmed action operation.

```
RORSapdu-example                ::= [2] IMPLICIT SEQUENCE {
    invokeID                    InvokeIDType,
    SEQUENCE {
    operation-value             INTEGER {m-Action-Confirmed (7)},
    result                      SEQUENCE {
    managedObjectClass          ObjectClass OPTIONAL,
    managedObjectInstance       ObjectInstance OPTIONAL,
    currentTime                 [5] IMPLICIT GeneralizedTime OPTIONAL,
    actionReply                 [6] IMPLICIT SEQUENCE (
    actionType                  ActionTypeId,
    actionReplyInfo             [4] ANY DEFINED BY actionType OPTIONAL } OPTIONAL
    }
```

-- ROIVapdu for the CMISE Linked Reply for a confirmed action operation.

```
ROIVapdu-linked-example         ::= [1] IMPLICIT SEQUENCE {
    invokeID                    InvokeIDType,
    linked-ID                   [0] IMPLICIT InvokeIDType,
    operation-value             INTEGER {m-Action-Confirmed (7)},
    argument                    CHOICE {
    actionResult                [4] IMPLICIT ActionResult,
    processingFailure           [5] IMPLICIT ProcessingFailure,
    actionError                 [7] IMPLICIT ActionError
    } }
```

-- ROERapdu for the CMISE confirmed action operation when a noSuchAction error occurs.

```
ROERapdu-example                ::= [3] IMPLICIT SEQUENCE {
    invokeID                    InvokeIDType,
    error-value                 INTEGER {noSuchAction (9)},
    parameter                   SEQUENCE { managedObjectClass   ObjectClass
                                                                OPTIONAL,
                                           actionId             ActionTypeId
    } }
```