

Recommendation X.407

MESSAGE HANDLING SYSTEMS ABSTRACT SERVICE DEFINITION CONVENTIONS 1

The establishment in various countries of telematic services and computer-based store-and-forward message services in association with public data networks creates a need to produce standards to facilitate international message exchange between subscribers to such services.

The CCITT,

considering

- (a) the need for Message Handling Systems;
 - (b) that Message Handling is a complex distributed information processing task;
 - (c) that a means for abstractly defining such tasks is required;
 - (d) that Recommendation X.200 defines the Reference Model of Open Systems Interconnection for CCITT applications;
 - (e) that Recommendations X.208, X.217, X.218, and X.219 provide the foundation for CCITT applications;
- unanimously declares
- (1) that conventions for defining abstract services are defined in section two;
 - (2) that techniques for the realization of abstract services so defined are discussed in section three.

Table of Contents

Section One - Introduction

- 0. Introduction
- 1. Scope
- 2. References
- 3. Definitions
- 4. Abbreviations
- 5. Conventions
- 5.1 ASN.1
- 5.2 Terms

Section TWO - Abstract Service Definition Conventions

- 6. Overview
- 7. Abstract Models
 - 7.1 Abstract Objects
 - 7.2 Abstract Ports
 - 7.3 Abstract Services
 - 7.4 Abstract Refinements
- 8. Abstract Services
 - 8.1 Abstract Procedures
 - 8.2 Abstract Bind Operations
 - 8.3 Abstract Unbind Operations
 - 8.4 Abstract Operations
 - 8.5 Abstract Errors

Section Three - Abstract Service Realizations

- 9. Overview
- 10. OSI Realizations

- 10.1 ROS Realizations
- 10.2 Non-ROS Realizations
- 11. Proprietary Realizations
- 11.1 Distributed Realizations
- 11.2 Non-distributed Realizations

Annexes

- A Example of Use of Abstract Service Notation
- A.1 Assignment of Object Identifiers
- A.2 Refinement of Yellow Environment
- A.3 Definition of Yellow Abstract Service
- A.4 Refinement of Green Environment
- A.5 Definition of Green Abstract Service
- A.6 Refinement of Yellow System
- A.7 Realization of Yellow System
- A.8 Realization of Green System
- B Reference Definition of Object Identifiers
- C Reference Definition of Notation
- D Differences Between CCITT Recommendation and ISO Standard
- E Index

CCITT Draft Recommendation X.407 MHS: Abstract Service Definition Conventions (Version 5, November 1987, Gloucester)

-- --

Section One - Introduction

0. Introduction

This Recommendation is one of a set of Recommendations for Message Handling. The entire set provides a comprehensive blueprint for a Message Handling System (MHS) realized by any number of cooperating open systems.

The Message Handling System made possible by these Recommendations is a complex distributed information processing task, many of whose components themselves have these characteristics.

This Recommendation specifies the conventions for defining the distributed information processing tasks of Message Handling and may also be useful for other applications.

The text of this Recommendation is the subject of joint CCITT-ISO agreement. The corresponding ISO specification is ISO 10021-4.

1. Scope

This Recommendation specifies the conventions used to specify the distributed information processing tasks that arise in Message Handling.

This Recommendation is structured as follows. Section one is this introduction. Section two specifies the conventions for defining a distributed information processing task abstractly. Section three gives principles for realizing the communication aspects of such tasks concretely, such as by Open Systems Interconnection (OSI) protocols. Annexes provide important supplemental information.

There are no requirements for conformance to this Recommendation.

2. References

This Recommendation cites the documents below.

- | | | |
|-------|--|------------------------------|
| X.200 | Basic reference model (see also ISO 7498). | |
| X.208 | Specification of abstract syntax notation one (ASN.1) (see also | ISO 8824). |
| X.209 | Specification of basic encoding rules for abstract syntax notation one | (ASN.1) (see also ISO 8825). |
| X.217 | Association control: Service definition (see also ISO 8649). | |

X.219 Remote operations: Model, notation and service definition (see also ISO 9072-1).

3. Definitions

For the purposes of this Recommendation, the definition of Annex E and below apply.

This Recommendation is based upon the concepts developed in Recommendation X.200 and uses the following terms defined in it:

- a) abstract syntax;
- b) Application Layer;
- c) application protocol data unit (.I.ab:APDU);
- d) application protocol;
- e) application service element (.I.ab:ASE);
- f) concrete transfer syntax;
- g) distributed information processing task;
- h) layer service;
- i) layer;
- j) open system;
- k) Open Systems Interconnection (.I.ab:OSI); and
- l) real open system.

This Recommendation uses the following terms defined in Recommendation X.208:

- a) Abstract Syntax Notation One (.I.ab:ASN.1);
- b) (data) type;
- c) (data) value;
- d) import;
- e) Integer;
- f) macro;
- g) module;
- h) Object Identifier; and
- i) tag.

This Recommendation uses the following terms defined in Recommendation X.209:

- a) Basic Encoding Rules.

This Recommendation uses the following terms defined in Recommendation X.217:

- a) application context (.I.ab:AC);

This Recommendation uses the following terms defined in Recommendation X.219:

- a) bind operation;
- b) error;
- c) linked;
- d) operation;
- e) Remote Operation Service (.I.ab:ROS);
- f) Remote Operations; and
- g) unbind operation.

4. Abbreviations

For the purposes of this Recommendation, the abbreviations of annex E apply.

5. Conventions

This Recommendation uses the descriptive conventions identified below.

5.1 ASN.1

This Recommendation uses for the indicated purposes the following ASN.1-based descriptive conventions.

- a) To define the OBJECT, PORT, and REFINE macros, the ASN.1 macro notation of Recommendation X.208.
- b) To define the ABSTRACT-BIND, -UNBIND, -OPERATION, and -ERROR macros, the BIND, UNBIND, OPERATION, and ERROR macros of Recommendation X.219.

- c) To specify the abstract syntax of information objects in the example of annex A, ASN.1 itself.
- d) To specify various abstract models in the example of annex A, the OBJECT, PORT, and REFINE macros of clause 7.
- e) To specify various abstract services in the example of annex A, the ABSTRACT-BIND, -OPERATION, and -ERROR macros of clause 8.

ASN.1 appears both in the body of this Recommendation to aid the exposition and again, largely redundantly, in annexes for reference. If differences are found between the two, a specification error is indicated.

Note that ASN.1 tags are implicit throughout the ASN.1 modules in the annexes; the modules are definitive in that respect.

5.2 Terms

Throughout this Recommendation, terms are rendered in bold when defined, in italic when referenced prior to their definitions, without emphasis upon all other occasions.

Terms that are proper nouns are capitalized, generic terms are not.

Section TWO - Abstract Service Definition Conventions

6. Overview

When faced with the job of describing and specifying a complex distributed information processing task, one is wise to begin by specifying the task in abstract, rather than concrete terms. This approach ensures that the task's functional requirements are stated independently of its concrete realization. Such separation is important, among other reasons, because each aspect of the task may admit of several concrete realizations. In a Message Transfer System comprising three message transfer agents, e.g., the first and second might interact using OSI communication, the second and third by proprietary means.

This section specifies the conventions for abstractly describing a distributed information processing task both macroscopically and microscopically. The former description is called an abstract model, the latter an abstract service.

Various formal tools for specifying abstract models and services are defined in this section. A comprehensive example of their use is given in annex A. The reader may wish to refer to that annex--e.g., to its illustrations--while reading the present section.

This section covers the following topics:

- a) Abstract models
- b) Abstract services

Note The formal tools mentioned above are neither a formal description language nor a substitute therefor. They are simply ASN.1 notation that supports the informal descriptive conventions this section defines.

7. Abstract Models

A macroscopic description of a distributed information processing task is called an **.I.gl:abstract model**; (**.I.gl:model**;) of that task and of the environment in which it is carried out. It is based upon the concepts of abstract objects, ports, services, and refinements. (The concept of an abstract service is much more fully developed in clause 8.)

7.1 Abstract Objects

An **.I.gl:abstract object**; (**.I.gl:object**;) is a functional entity, one of perhaps several which interact with one another. Objects are of different types which determine their function and behavior. An object of one type, e.g., might represent a system, multiple objects of another type its users. Objects interact by means of abstract ports.

An object type is specified by means of the OBJECT macro. Such a specification lists the types of abstract ports that provide access to such an object. For each assymetric port type, the specification indicates whether the ports of that type are consumer or supplier ports.

```
.I.ma:OBJECT; MACRO ::= BEGIN TYPE NOTATION ::= "PORTS" "{" PortList "}" | empty VALUE NOTATION ::=
value (VALUE OBJECT IDENTIFIER) PortList ::= Port "," PortList | Port Port ::= value (PORT) PortType
PortType ::= Symmetric | Asymmetric Symmetric ::= empty Asymmetric ::= Consumer | Supplier
Consumer ::= "[C]" Supplier ::= "[S]" END
```

A data value of type OBJECT is an Object Identifier that unambiguously and uniquely identifies the specified object type.

Note - The keyword "OBJECT" is reserved in ASN.1. Selection of a suitable replacement for use in the present context is **for further study**.

7.2 Abstract Ports

An abstract port (port) is a point at which an abstract object interacts with another abstract object. Ports are of different types which determine the kinds of interactions they enable. Ports of one type, e.g., might represent the means by which a directory system is accessed, ports of another type the means by which it is administered. Port types are themselves of the following two varieties:

- a) symmetric: All instances of a symmetric port type are identical.
- b) asymmetric: Each instance of an asymmetric port type is of one of two kinds, supplier and consumer.

Note - A particular allocation of the terms "supplier" and "consumer" is often intuitive. One might naturally consider a file system, e.g., to present supplier ports to its users and administrators. Strictly speaking, however, the assignment of the two terms is arbitrary.

Two objects can interact with one another by means of a port in one and a port in the other only while those ports are in contact with one another, or bound. The actions by means of which this state is initiated and terminated for one or more port pairs are called binding and unbinding, respectively.

Two ports can be bound only if they match. Any two ports of the same, symmetric type match. Two ports of the same, asymmetric type match if and only if one is a supplier, the other a consumer.

A port type is specified by means of the PORT macro. Such a specification identifies the abstract operations that represent the interactions possible while two such ports are bound. If none is listed, the abstract operations shall be considered unspecified.

```
.I.ma:PORT; MACRO ::= BEGIN TYPE NOTATION ::= Operations | empty VALUE NOTATION ::= value (VALUE
OBJECT IDENTIFIER) Operations ::= Symmetrical | Asymmetrical Symmetrical ::= "ABSTRACT"
"OPERATIONS" "{" OperationList "}" Asymmetrical ::= OneSided | TwoSided OneSided ::= Consumer |
Supplier TwoSided ::= Consumer Supplier | Supplier Consumer Consumer ::= "CONSUMER" "INVOKES"
"{" OperationList "}" Supplier ::= "SUPPLIER" "INVOKES" "{" OperationList "}" OperationList ::= Operation
"," OperationList | Operation Operation ::= value (ABSTRACT-OPERATION) END
```

If the port type is symmetric, both objects offer all listed abstract operations. If the port type is asymmetric, the macro distinguishes between the abstract operations an object with the consumer port offers and those an object with the supplier port offers.

A data value of type PORT is an Object Identifier that unambiguously and uniquely identifies the specified port type.

7.3 Abstract Services

An abstract service is the set of capabilities that one object offers to another by means of one or more of its ports. The former object is called an abstract service provider (provider), the latter an abstract service user (user). Each port in question may be either symmetric or asymmetric and, if the latter, either consumer or supplier.

An abstract service may have any number of users and providers.

Whenever the abstract service ports of a provider are bound to the matching ports of a user, an **abstract association** (or **association**) is said to exist between the two objects.

An abstract service is specified as indicated in clause 8.

Note - An abstract service serves much the same purpose within the Application Layer as does one of the layer services of lower OSI layers.

7.4 Abstract Refinements

An object can be viewed in different ways at different times. On some occasions it is convenient to think of an object as atomic. This is the case, e.g., when describing how an object interacts with other objects external to it, i.e., when specifying its abstract service. On other occasions, it may be more convenient to think of an object as composite, i.e., constructed from other objects. This might be the case, e.g., when describing how an object is realized.

Like any objects, component objects have ports. Some are those visible on the "surface" of the constructed object. Others enable the component objects to interact, thus supporting the provision and use of lesser abstract services among the component objects, which cooperate to provide the overall abstract service of the constructed object.

The functional decomposition of an object into several lesser objects is called the abstract refinement (refinement) of that object.

The technique of refinement can be applied recursively. A component object can itself be refined to reveal its internal structure. This can continue until one reaches component objects best considered atomic.

A refinement is specified by means of the REFINE macro. It identifies the object whose internal structure is being revealed and the component objects used in its construction. Each component object is characterized as either unique or recurring. The macro also indicates which ports of component objects are bound to ports of other component objects, and which are visible at the surface of the composite object.

```
.I.ma:REFINE; MACRO ::= BEGIN TYPE NOTATION ::= Object "AS" ComponentList VALUE NOTATION ::=
value (VALUE OBJECT IDENTIFIER) ComponentList ::= Component ComponentList | Component
Component ::= ObjectSpec Ports ObjectSpec ::= Object | Object "RECURRING" Ports ::= PortSpecList |
empty PortSpecList ::= PortSpec PortSpecList | PortSpec PortSpec ::= value (PORT) PortSide PortStatus PortSide
::= Consumer | Supplier | empty Consumer ::= "[C]" Supplier ::= "[S]" PortStatus ::= "VISIBLE" |
"PAIRED" "WITH" ObjectList ObjectList ::= Object "," ObjectList | Object Object ::= value (OBJECT) END
```

A data value of type REFINE is an Object Identifier.

Note As with objects themselves, ports, in principle, can be viewed in different ways at different times. On some occasions, it is convenient to think of a port (pair) as atomic. However, one can imagine refining a port itself to examine how communication of this type can be provided. In this view, a port pair is itself viewed as being supported by a collection of objects. This would enhance the ability to specify communications capabilities. This "port refinement" concept is not pursued further in this version of this Recommendation.

8. Abstract Services

A microscopic description of a distributed information processing task is a specification of the abstract service that defines how the task is initiated, controlled, and terminated. It is based upon the concepts of abstract bind operations, unbind operations, operations, and errors, as well as the enabling concept of abstract procedures.

Note The macros defined below imply use of ASN.1 to specify arguments, results, and parameters. Any context-specific tags, e.g., assigned in the course of the specifications, although meaningless in that context, play an important role in a ROS realization of the abstract service.

8.1 Abstract Procedures

An .I.gl:abstract procedure; (.I.gl:procedure;) is a task that one object carries out at another's request. The making of the request and the carrying out of the task are called the .I.gl:invocation; and .I.gl:performance; of the procedure. The objects that issue and act upon the request are called the .I.gl:invoker; and .I.gl:performer;; respectively.

A procedure may (but need not) require that an invoker, upon invocation, supply to the performer a single information object of a prescribed type, which is called the procedure's .I.gl:argument;.

Every performance of every procedure has an outcome, success or failure. A procedure is considered to succeed if it is carried out in full, to fail if it is terminated prematurely.

A procedure may (but need not) require that the performer apprise the invoker of success. It may (but need not) further require that it supply, when reporting success, a single information object of a prescribed type, which is called the procedure's .I.gl:result;.

A procedure may (but need not) require that the performer apprise the invoker of failure. It may (but need not) further require that it supply certain information when reporting failure.

Note In subsequent clauses ASN.1 is prescribed as the means for specifying the abstract syntax of the arguments and results of procedures (as well as of the parameters of abstract errors). These uses of ASN.1 do not imply that these information objects are necessarily transported between open systems. In particular, the fact that the information objects, by virtue of their description in ASN.1 and of its Basic Encoding Rules, have concrete transfer syntaxes is immaterial in the present context. ASN.1 is simply a convenient tool for formally describing the information objects' abstract syntax.

8.2 Abstract Bind Operations

An .I.gl:abstract bind operation; is a procedure whose successful performance binds one or more pairs of abstract ports. The object which invokes an abstract bind operation is said to be the .I.gl:initiator;, that which performs it the .I.gl:responder;.

An abstract bind operation is suitable for binding a particular set of ports of the initiator to a matching set of the responder. Where one or more ports in the set are asymmetric, the abstract bind operation may be suitable for binding to the consumer side only, the supplier side only, or to either.

An abstract bind operation is a fully general procedure except that, if information is conveyed to the invoker upon failure, it is constrained to a single information object, called .I.gl:error information;.

An abstract bind operation is specified by means of the ABSTRACT-BIND macro whose definition is as follows:

```
.I.ma:ABSTRACT-BIND; MACRO ::= BEGIN TYPE NOTATION ::= Ports Bind VALUE NOTATION ::= value
(VALUE BindType) Ports ::= "TO" "{" PortList "}" | empty PortList ::= Port "," PortList | Port Port ::=
value (PORT) PortSide PortSide ::= Consumer | Supplier | empty Consumer ::= "[C]" Supplier ::= "[S]"
Bind ::= type (BindType) -- must be a BIND type | empty <BindType ::= BIND> END
```

The "Ports" clause, introduced by the keyword "TO", lists the ports of a responder which this abstract bind operation will bind. If an asymmetric port is listed there, without being qualified by "[S]" or "[C]", this means that the abstract bind operation is suitable for use in binding such a port in either direction.

Note that the specification of the argument, result, and/or error information is accomplished by means of an (embedded) BIND macro of Remote Operations, defined in Recommendation X.219, and it is a value of such a type that the macro returns. If none is provided, the default "BIND" is returned.

Note The relationship of ABSTRACT-BIND and BIND can help make trivial the ROS realization of an abstract service; see clause 10.1.

An abstract service typically comprises an abstract bind operation for each type of port involved in its provision. When several port types are involved, their abstract bind operations may but need not be distinct.

8.3 Abstract Unbind Operations

An .I.gl:abstract unbind operation; is a procedure whose performance, successful or not, unbinds two ports. It is invoked by the object which invoked the corresponding abstract bind (i.e., the initiator) and performed by the responder.

An abstract unbind operation is suitable for unbinding a particular set of ports of the initiator from a matching set of the responder. Where one or more ports in the set are asymmetric, the abstract unbind operation may be suitable for unbinding from the consumer side only, the supplier side only, or either.

An abstract unbind operation is a fully general procedure except that, if information is conveyed to the invoker upon failure, it is constrained to a single information object, called .I.gl:error information;.

An abstract unbind operation is specified by means of the ABSTRACT-UNBIND macro whose definition is as follows:

```
.I.ma:ABSTRACT-UNBIND; MACRO ::= BEGIN TYPE NOTATION ::= Ports Unbind VALUE NOTATION ::=
value (VALUE UnbindType) Ports ::= "FROM" "{" PortList "}" PortList ::= Port "," PortList | Port Port
```

```

::= value (PORT) PortSide PortSide      ::= Consumer | Supplier | empty Consumer      ::= "[C]" Supplier      ::= "[S]"
Unbind      ::= type (UnbindType) |      -- must be an UNBIND type      empty <UnbindType ::=
UNBIND> END

```

The "Ports" clause, introduced by the keyword "FROM", lists the ports of a responder from which this abstract unbind operation will unbind. If an asymmetric port is listed there, without being qualified by "[S]" or "[C]", this means that the abstract unbind operation is suitable for use in unbinding such a port in either direction (although the actual direction is determined by the direction in which the bind took place).

Note that the specification of the argument, result, and/or error information is accomplished by means of an (embedded) UNBIND macro of Remote Operations, defined in Recommendation X.219, and it is a value of such a type that the macro returns. If none is provided, the default "UNBIND" is returned.

Note The relationship of ABSTRACT-UNBIND and UNBIND helps make trivial the ROS realization of an abstract service; see clause 10.1.

An abstract service typically comprises an abstract unbind operation for each type of port involved in its provision. When several port types are involved, their abstract unbind operations may but need not be distinct.

8.4 Abstract Operations

An .I.gl:abstract operation; is a procedure that may be invoked in the context of two bound ports. Its failure has no effect upon the binding. If the ports are asymmetric, whether the invoker is the object having the consumer port, the object having the supplier port, or either is prescribed by the port. If the ports are symmetric, the invoker may be either object. Whether the ports are symmetric or asymmetric, the remaining object is the performer.

An abstract operation is a fully general procedure except for the information conveyed to the invoker upon failure. An abstract operation fails when it encounters an abstract error, and the information conveyed is constrained to that required to report that abstract error. Whether failure is reported and, if so, which abstract errors can be encountered are prescribed for each abstract operation.

An abstract operation is specified by means of the ABSTRACT-OPERATION macro. Its definition is identical to that of the OPERATION macro of Remote Operations, specified in Recommendation X.219.

```
.I.ma:ABSTRACT-OPERATION; MACRO ::= OPERATION
```

An abstract service comprises zero or more abstract operations for each type of port involved in its provision. When several port types are involved, they may but need not have abstract operations in common.

Note The equivalence of ABSTRACT-OPERATION and OPERATION helps make trivial the ROS realization of an abstract service; see clause 10.1.

8.5 Abstract Errors

An .I.gl:abstract error; is an exceptional condition that may arise during the performance of an abstract operation, causing it to fail.

When an abstract error is reported, the performer conveys to the invoker the identity of the abstract error and possibly a single information object called its .I.gl:parameter;. Whether a parameter is returned and, if so, its type are prescribed for each abstract error.

An abstract error is specified by means of the ABSTRACT-ERROR macro. Its definition is identical to that of the ERROR macro of Remote Operations, specified in Recommendation X.219.

```
.I.ma:ABSTRACT-ERROR; MACRO ::= ERROR
```

An abstract service comprises the zero or more abstract errors reported by its abstract operations.

Note The equivalence of ABSTRACT-ERROR and ERROR helps make trivial the ROS realization of an abstract service; see clause 10.1.

Section Three - Abstract Service Realizations

9. Overview

Once a distributed information processing task has been described and specified in abstract terms, the manner in which each aspect of the task is to be concretely realized must be prescribed. As suggested previously, each aspect may admit of several concrete realizations.

This section specifies principles for concretely realizing abstract models and services. A .I.gl:real; x is the computer process or system, or the real open system that concretely realizes an abstract object of type x.

This section covers the following topics:

- a) OSI realizations
- b) Proprietary realizations

Note The aspects of an abstract model stressed here are abstract ports and their bindings. This is because abstract ports mark the boundary not only between abstract objects but also between the physical systems that concretely realize those abstract objects. Thus abstract ports and bindings are the parts of an abstract model that must be constructed or constructable with OSI tools if open systems interworking is to occur.

10. OSI Realizations

A primary objective of CCITT Recommendations and ISO Standards is to specify how distributed information processing tasks are realized when carried out by several cooperating real open systems.

In the OSI environment, objects are realized by means of application processes, with, in general, a many-to-many mapping of objects to application processes. Communication among objects which are realized by application processes in different open systems is accomplished by OSI application protocols (consisting of application contexts). An application context thus realizes the binding, use, and unbinding of a number of port pairs.

The specification of an application context is in terms of the coordinated operation of a number of application-service-elements. Realization is therefore particularly straightforward to specify if an application-service-element is defined to correspond to each port whose communication is to be supported.

The realization of abstract ports and bindings by means of ASEs and ACs is discussed below. Both ROS and non-ROS realizations are considered.

10.1 ROS Realizations

The concrete realization of ports and bindings is often trivial when accomplished by means of Remote Operations.

This is true because it is straightforward to define an abstract service which is such that there exists a ROS-based application protocol that is functionally identical to it. This is true in turn because the framework for the specification of abstract services is isomorphic to that for the specification of ROS-based application protocols. The correspondences behind the isomorphism are listed in Table 1/X.407.

Table .T.:1/X.407 Correspondences of Abstract Services and ROS-based Protocols

+-----+-----+ Aspect of		Aspect of	Abstract Service		ROS-based
Protocol +-----+-----+ Abstract bind operation	Bind operation	Abstract unbind			
operation Unbind operation	Abstract operation	Operation	Abstract error	Error	
+-----+-----+					

The correspondences of the table arise from the fact that corresponding aspects are formally specified using closely-related, or equivalent macros, as summarized in Table 2/X.407:

Table .T.:2/X.407 Equivalent Abstract Service and ROS Macros

+-----+-----+ Abstract Service Macro ROS Macro +-----+-----+				
ABSTRACT-BIND	BIND	ABSTRACT-UNBIND	UNBIND	ABSTRACT-OPERATION
OPERATION	ABSTRACT-ERROR	ERROR	+-----+-----+	

The definition of ROS-based ASEs and ACs that concretely realize abstract ports is explored in annex A by means of an example.

For the realization to be trivial, it is necessary that there be an abstract bind operation which binds all of the ports which must be paired.

Note Where there is more than one port (pair) involved in the abstract service, this requires that the abstract bind operation be designed for the particular ports involved. There is (currently) no provision for the automatic synthesis of a suitable abstract bind based, e.g., upon the definitions of abstract bind operations defined for the individual ports.

10.2 Non-ROS Realizations

The concrete realization of ports and bindings is a more substantial task when attempted by means other than Remote Operations, and little can be said about the general proposition.

Despite the above, the following two observations are relevant:

- a) The concrete realization of an abstract service as an application protocol is greatly simplified by using ASN.1 to define its APDUs. This is so because the protocol specification can simply import relevant types and values from the abstract service specification.
- b) The concrete realization of an abstract service whose abstract operations do not report their outcomes is conceptually simple. This is so because each such abstract operation represents an interaction comprising a single

APDU. From this simplest of all possible interactions, arbitrarily complex ones can be constructed.

11. Proprietary Realizations

A secondary objective of CCITT Recommendations and ISO Standards is to ensure that those portions of a distributed information processing task that are carried out by proprietary means are accomplished in such a way that the intended overall functionality of the system is upheld.

The realization of abstract ports and bindings by proprietary means is briefly discussed below. Both distributed and non-distributed realizations are considered.

11.1 Distributed Realizations

The concrete realization of ports and bindings by means of proprietary computer communication protocols is a local matter. The specification of the visible functionality embodied in the abstract service provides a guide to the implementors of the proprietary realizations, so that, where such realizations are appropriate, they may play the appropriate role in the overall task.

11.2 Non-distributed Realizations

The concrete realization of ports and bindings by means of mechanisms wholly within a single computer is a local matter. As with the case considered in clause 11.1, the abstract service specification serves as a guide to the implementor in ensuring that the proprietary realization can nonetheless play the appropriate role in the overall task.

Annexes

Annex A (to Recommendation X.407) Example of Use of Abstract Service Notation

This annex is not a part of this Recommendation.

This annex illustrates the use of the abstract model and service notation by means of an example. The example involves two systems, the Yellow and Green Systems, and their environments, the Yellow and Green Environments.

It uses the abstract model notation to describe the environments separately (clauses A.2 and A.4) and to show how their systems are related: one is constructed from the other (clause A.6). It uses the abstract service notation to describe the capabilities of each system (clause A.3 and A.5). The example concludes by realizing the systems' ports as ACs and ASEs using the ROS notation of Recommendation X.219, as might be appropriate for OSI communication (clause A.7 and A.8).

A.1 Assignment of Object Identifiers

The ASN.1 modules defined in this annex require the assignment of a variety of Object Identifiers. All are defined below using ASN.1. The assignments are definitive except for those for ASN.1 modules and the subject of application service definition conventions itself. The definitive assignments for the former occur in the modules themselves; other references to them appear in IMPORT clauses. The latter is fixed.

```
-----
ExampleObjectIdentifiers {joint-iso-ccitt                                mhs(6) asdc(2) example(1) modules(0)
object-identifiers(0)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue -- Exports everything. IMPORTS --
nothing -- ; ID ::= OBJECT IDENTIFIER -- Abstract Service Definition Conventions Example (not definitive) id-
asdc-ex ID ::= {joint-iso-ccitt mhs(6) asdc(2) example(1)} -- not definitive -- Categories id-mod ID ::= {id-asdc-ex 0}
-- modules; not definitive id-ot ID ::= {id-asdc-ex 1} -- object types id-pt ID ::= {id-asdc-ex 2} -- port types id-ref
ID ::= {id-asdc-ex 3} -- refinements id-ac ID ::= {id-asdc-ex 4} -- application contexts id-ase ID ::= {id-asdc-ex 5} --
application service elements id-as ID ::= {id-asdc-ex 6} --- abstract syntaxes -- Modules id-mod-object-identifiers
ID ::= {id-mod 0} -- not definitive id-mod-ye-refinement ID ::= {id-mod 1} -- not definitive id-mod-y-abstract-
service ID ::= {id-mod 2} -- not definitive id-mod-ge-refinement ID ::= {id-mod 3} -- not definitive id-mod-g-
abstract-service ID ::= {id-mod 4} -- not definitive id-mod-ys-refinement ID ::= {id-mod 5} -- not definitive id-
mod-ys-realization ID ::= {id-mod 6} -- not definitive id-mod-gs-realization ID ::= {id-mod 7} -- not definitive --
Object types id-ot-y-environment ID ::= {id-ot 0} id-ot-y-user ID ::= {id-ot 1} id-ot-y-system ID ::= {id-ot 2}
id-ot-g-environment ID ::= {id-ot 3} id-ot-g-user ID ::= {id-ot 4} id-ot-g-manager ID ::= {id-ot 5} id-ot-g-
system ID ::= {id-ot 6} id-ot-agent ID ::= {id-ot 7} -- Port types id-pt-y-use ID ::= {id-pt 0} id-pt-g-use
ID ::= {id-pt 1} id-pt-g-management ID ::= {id-pt 2} -- Refinements id-ref-y-environment ID ::= {id-ref 0} id-ref-g-
environment ID ::= {id-ref 1} id-ref-y-system ID ::= {id-ref 2} -- Application contexts id-ac-y-use ID ::= {id-
ac 0} id-ac-g-use ID ::= {id-ac 1} id-ac-g-management ID ::= {id-ac 2} -- Application service elements id-ase-y-
use ID ::= {id-ase 0} id-ase-g-use ID ::= {id-ase 1} id-ase-g-management ID ::= {id-ase 2} -- Abstract
```

```

syntaxes id-as-y-use      ID ::= {id-as 0} id-as-g-use      ID ::= {id-as 1} id-as-g-management ID ::= {id-as 2} END
-- of ExampleObjectIdentifiers

```

A.2 Refinement of Yellow Environment

The Yellow Environment, depicted in Figure 1/X.407, is formally refined below using the OBJECT and REFINE macros.

```

+----+ | 01 || 02 || 03 || 04 || 05 || 06 || 07 || 08 || 09 || 10 || 11 || 12 || 13 || 14 || 15 || 16 || 17 || 18 || 19 || 20 || 21 |
| 22 || 23 || 24 || 25 || 26 || 27 || 28 || 29 | +----+

```

Figure .F:1/X.407 The Yellow Environment

As the figure indicates and the ASN.1 specification below confirms, the Yellow Environment can be modeled as an object which can be decomposed into one central object, the Yellow System, and any number of other, peripheral objects, yellow users. The Yellow System interacts with yellow users by means of its yellow-use ports.

```

-----
YellowEnvironmentRefinement {joint-iso-ccitt                                mhs(6) asdc(2) example(1) modules(0) ye-
refinement(1)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS    yellow-environment, yellow-
environment-refinement,
yellow-system, yellow-user; IMPORTS                                       -- Yellow Abstract Service
yellow-use      ---- FROM YellowAbstractService {joint-iso-ccitt          mhs(6) asdc(2) example(1) modules(0) y-
abstract-service(2)}              -- Example Object Identifiers
id-ot-y-environment, id-ot-y-system, id-ot-y-user,

id-ref-y-environment              ---- FROM ExampleObjectIdentifiers {joint-
iso-ccitt    mhs(6) asdc(2) example(1) modules(0) object-identifiers(0)} -- Abstract Service Notation
OBJECT, REFINE                    ---- FROM AbstractServiceNotation {joint-iso-ccitt    mhs(6) asdc(2) modules(0)
notation(1)}; -- Yellow Environment yellow-environment OBJECT ::= id-ot-y-environment -- Yellow Environment
refinement yellow-environment-refinement REFINE yellow-environment AS yellow-user RECURRING
yellow-system                    yellow-use [S] PAIRED WITH {yellow-user} ::= id-ref-y-environment -- Component
object types yellow-user OBJECT PORTS { yellow-use [C]}                  ::= id-ot-y-user yellow-system OBJECT PORTS
{ yellow-use [S]}
::= id-ot-y-system END -- of YellowEnvironmentRefinement

```

A.3 Definition of Yellow Abstract Service

The abstract service that the Yellow System provides to its users is formally defined below using the PORT and ABSTRACT-BIND, -OPERATION, and -ERROR macros.

As the ASN.1 specification indicates, the abstract service that the Yellow System provides comprises ports of a single kind, yellow-use. Each port comprises a number of abstract operations which collectively report a number of abstract errors. The Yellow System guards its ports by means of an abstract bind operation, YellowBind, which demands that users identify themselves convincingly before further interaction occurs. An abstract unbind operation, YellowUnbind, which constitutes the finalization step required to conclude an interaction.

```

-----
YellowAbstractService {joint-iso-ccitt                                mhs(6) asdc(2) example(1) modules(0) y-abstract-
service(2)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS
AuthenticateUser, Yellow-operation-1, ... yellow-use; IMPORTS              -- Example Object Identifiers id-pt-y-use ---- FROM
ExampleObjectIdentifiers {joint-iso-ccitt                                mhs(6) asdc(2) example(1) modules(0) object-
identifiers(0)} -- Abstract Service Notation
ABSTRACT-BIND, ABSTRACT-ERROR, ABSTRACT-OPERATION, PORT ---- FROM AbstractServiceNotation {joint-iso-
ccitt    mhs(6) asdc(2) modules(0) notation(1)}; -- Port type yellow-use PORT    CONSUMER INVOKES { Yellow-
operation-1, ...} ::= id-pt-y-use -- Abstract bind operation Credentials ::= SET {      name    [0] IA5String,
password [1] IA5String} YellowBind ::= ABSTRACT-BIND    TO {yellow-use[S]}    BIND    ARGUMENT
credentials Credentials    BIND-ERROR ENUMERATED {      name-or-password-invalid(0)} --
Abstract unbind operation YellowUnbind ::= ABSTRACT-UNBIND    FROM {yellow-use[S]} -- Abstract operations
Yellow-operation-1 ::= ABSTRACT-OPERATION ARGUMENT ... RESULT ...    ERRORS {      yellow-error-1, ...} ...
-- Abstract errors yellow-error-1 ABSTRACT-ERROR    PARAMETER ... ::= 1 ... END -- of

```

YellowAbstractService

A.4 Refinement of Green Environment

The Green Environment, depicted in Figure 2/X.407, is formally refined below using the OBJECT and REFINE macros.

+----+ | 01 || 02 || 03 || 04 || 05 || 06 || 07 || 08 || 09 || 10 || 11 || 12 || 13 || 14 || 15 || 16 || 17 || 18 || 19 || 20 || 21 || 22 ||
23 || 24 || 25 || 26 || 27 || 28 || 29 || +----+

Figure .F.:2/X.407 The Green Environment

As the figure indicates and the ASN.1 specification below confirms, the Green Environment can be modeled as an object which can be decomposed into one central object, the Green System; any number of other, peripheral objects, green users; and any number of yet additional objects, green managers. The Green System interacts with green users and managers by means of its green-use ports, and with green managers (alone) by means of its green-management ports.

```
-----
GreenEnvironmentRefinement {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) ge-
refinement(3)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS green-environment, green-
environment-refinement, green-manager, green-system, green-user; IMPORTS -- Green Abstract Service green-
use, green-management ---- FROM GreenAbstractService {joint-iso-ccitt mhs(6) asdc(2) example(1)
modules(0) g-abstract-service(4)} -- Example Object Identifiers id-ot-g-environment, id-ot-g-manager, id-ref-g-
environment id-ot-g-user, id-ot-g-system, ---- FROM ExampleObjectIdentifiers {joint-iso-
ccitt mhs(6) asdc(2) example(1) modules(0) object-identifiers(0)} -- Abstract Service Notation OBJECT,
REFINE ---- FROM AbstractServiceNotation {joint-iso-ccitt mhs(6) asdc(2) modules(0) notation(1)}; -- Green
Environment green-environment OBJECT ::= id-ot-g-environment -- Green Environment refinement green-environment-
refinement REFINE green-environment AS
green-user RECURRING green-manager RECURRING green-system green-use [S] PAIRED WITH
{green-user, green-manager} green-management [S] PAIRED WITH {green-manager}
::= id-ref-g-environment -- Component object types green-user OBJECT PORTS { green-use [C]} ::= id-ot-g-user
green-manager OBJECT PORTS { green-use [C], green-management [C]}
::= id-ot-g-manager green-system OBJECT PORTS { green-use [S], green-
management [S]} ::= id-ot-g-system END -- of GreenEnvironmentRefinement
```

A.5 Definition of Green Abstract Service

The abstract service that the Green System provides to its users and managers is formally defined below using the PORT and ABSTRACT-BIND, -OPERATION, and -ERROR macros.

As the ASN.1 specification indicates, the abstract service that the Green System provides comprises ports of two kinds, green-use and green-management. A port of either kind comprises a number of abstract operations which collectively report a number of abstract errors. The Green System guards its ports by means of abstract bind operations, AuthenticateUser and AuthenticateManager, which demand that users and managers identify themselves convincingly before further interaction can occur. No abstract unbind operations are specified, indicating that no finalization step is required to conclude an interaction.

```
-----
GreenAbstractService {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) g-abstract-
service(4)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS
AuthenticateManager, AuthenticateUser, green-management, Green-management-operation-1, ... green-use, Green-
use-operation-1, ...; IMPORTS -- Example Object Identifiers id-pt-g-use, id-pt-g-management ---- FROM
ExampleObjectIdentifiers {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) object-
identifiers(0)} -- Abstract Service Notation PORT, ABSTRACT-BIND, ABSTRACT-
OPERATION, ABSTRACT-ERROR ---- FROM AbstractServiceNotation {joint-iso-ccitt mhs(6) asdc(2) modules(0)
notation(1)}; -- Port types green-use PORT CONSUMER INVOKES { Green-use-
operation-1, ...} ::= id-pt-g-use green-management PORT CONSUMER INVOKES { green-management-
operation-1, ...} ::= id-pt-g-management -- Abstract bind operations Credentials ::= SET { name [0] IA5String,
password [1] IA5String} AuthenticateUser ::= ABSTRACT-BIND ARGUMENT credentials Credentials BIND-
ERROR ENUMERATED { name-or-password-invalid(0)}
AuthenticateManager ::= ABSTRACT-BIND ARGUMENT credentials Credentials BIND-ERROR ENUMERATED {
name-or-password-invalid(0), not-a-manager (1)} -- Abstract operations
Green-use-operation-1 ::= ABSTRACT-OPERATION ARGUMENT ... RESULT ... ERRORS { green-error-1, ...} ...
```

```

Green-management-operation-1 ::= ABSTRACT-OPERATION ARGUMENT ... RESULT ... ERRORS {
green-error-1, ...} ... -- Abstract errors green-error-1 ABSTRACT-ERROR PARAMETER ... ::= 1 ... END -- of
GreenAbstractService

```

A.6 Refinement of Yellow System

The Yellow System, depicted in Figure 3/X.407, is formally refined below using the OBJECT and REFINE macros.

```

+----+ | 01 | | 02 | | 03 | | 04 | | 05 | | 06 | | 07 | | 08 | | 09 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | 16 | | 17 | | 18 | | 19 | | 20 | | 21 | | 22 | |
23 | | 24 | | 25 | | 26 | | 27 | | 28 | | 29 | | +----+

```

Figure .F.:3/X.407 The Yellow System

As the figure indicates and the ASN.1 specification confirms, the Yellow System, when examined closely, has components. In particular, the Yellow System comprises the Green System and green managers, augmented by objects of an as yet unseen variety, agent. An agent serves as an intermediary between the Green System and a yellow user. It might be thought of as adding value to the Green System. In any case, it is a provider of a yellow-use port and a consumer of a green-use port.

```

YellowSystemRefinement {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) ys-
refinement(5)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS agent, yellow-system-refinement;
IMPORTS -- Yellow Environment Refinement yellow-system, yellow-use ---- FROM
YellowEnvironmentRefinement {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) ye-
refinement(1)} -- Green Environment Refinement green-management, green-manager, green-system,
green-use ---- FROM GreenEnvironmentRefinement {joint-iso-ccitt mhs(6) asdc(2) example(1)
modules(0) ge-refinement(3)} -- Example Object Identifiers id-ot-agent, id-ref-y-system ---- FROM
ExampleObjectIdentifiers {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) object-
identifiers(0)} -- Abstract Service Notation OBJECT, REFINE FROM
AbstractServiceNotation {joint-iso-ccitt mhs(6) asdc(2) modules(0) notation(1)}; -- Yellow
System refinement yellow-system-refinement REFINE yellow-system AS agent RECURRING yellow-use [S]
VISIBLE green-manager RECURRING green-system green-use [S] PAIRED WITH
{agent, green-manager} green-management [S] PAIRED WITH {green-manager} ::= id-ref-y-system --
Component object type agent OBJECT PORTS { yellow-use [S], green-use
[C]} ::= id-ot-agent END -- of YellowSystemRefinement

```

A.7 Realization of Yellow System

The abstract service of the Yellow System is formally realized below, by means of ROS, using the APPLICATION-CONTEXT and APPLICATION-SERVICE-ELEMENT macros of Recommendation X.219.

As the ASN.1 specification indicates, the abstract service that the Yellow System provides is realized as a single ASE, yellow-use-ASE, and a single and corresponding AC, yellow-use-AC. Each abstract bind operation, abstract operation, or abstract error in the abstract service has a corresponding and equivalent bind operation, operation, or error, respectively, in its ROS-based realization.

Note that Integer values are assigned to the operations; the corresponding abstract operations require and received no such values.

```

YellowSystemRealization {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) ys-
realization(6)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS yellow-use-AC, yellow-use-ASE;
IMPORTS -- Yellow Abstract Service Yellow-operation-1, ... yellow-use, YellowBind,
YellowUnbind ---- FROM YellowAbstractService {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) y-
abstract-service(2)} -- Example Object Identifiers id-ac-y-use, id-as-y-use, id-ase-y-use ---- FROM
ExampleObjectIdentifiers {joint-iso-ccitt mhs(6) asdc(2) example(1) modules(0) object-
identifiers(0)} -- Remote Operations APDUs rOSE ---- FROM Remote-Operations-
APDUs {joint-iso-ccitt remote-operations(4) apdus(1)} -- Association Control aCSE, aCSE-AS ---- FROM
AssociationControl {joint-iso-ccitt -- Remote Operations Notation Extension
acse(2) .I.bs:Module imports;to be supplied} -- Remote Operations Notation Extension
APPLICATION-CONTEXT, APPLICATION-SERVICE-ELEMENT ---- FROM Remote-Operations-Notation-extension
{joint-iso-ccitt remote-operations(4) notation-extension(2)}; -- Application context yellow-use-AC APPLICATION-
CONTEXT APPLICATION SERVICE ELEMENTS {aCSE} BIND YellowBind

```

```

UNBIND YellowUnbind          REMOTE OPERATIONS {rOSE}  INITIATOR CONSUMER OF {yellow-use-ASE}
    ABSTRACT SYNTAXES {yellow-use-AS, aCSE-AS} ::= id-ac-y-use -- Application service element yellow-use-ASE
APPLICATION-SERVICE-ELEMENT          CONSUMER INVOKES {    yellow-
operation-1, ...} ::= id-ase-y-use yellow-operation-1 Yellow-operation-1 ::= 1 ... -- Abstract syntax yellow-use-AS OBJECT
IDENTIFIER ::= id-as-y-use END -- of YellowSystemRealization

```

A.8 Realization of Green System

The abstract service of the Green System is formally realized below, by means of ROS, using the APPLICATION-CONTEXT and APPLICATION-SERVICE-ELEMENT macros of Recommendation X.219.

As the ASN.1 specification indicates, the abstract service that the Green System provides is realized as two ASEs, green-use-ASE and green-management-ASE, and two, corresponding ACs, green-use-AC and green-management-AC. Each abstract bind operation, abstract operation, or abstract error in the abstract service has a corresponding and equivalent bind operation, operation, or error, respectively, in its ROS-based realization.

Note that Integer values are assigned to the operations; the corresponding abstract operations require and received no such values.

```

GreenSystemRealization {joint-iso-ccitt          mhs(6) asdc(2) example(1) modules(0) gs-
realization(7)} DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS    green-management-AC, green-
management-ASE, green-use-AC, green-use-ASE; IMPORTS          -- Green Abstract Service
    AuthenticateManager, AuthenticateUser,          green-management, Green-management-
operation-1, ...    green-use, Green-use-operation-1, ...      ---- FROM GreenAbstractService {joint-iso-ccitt
    mhs(6) asdc(2) example(1) modules(0) g-abstract-service(4)} -- Example Object Identifiers          id-ac-g-use,
id-ase-g-use,      id-as-g-use,          id-ac-g-management, id-ase-g-management, id-as-g-
management          ---- FROM ExampleObjectIdentifiers {joint-iso-ccitt mhs(6) asdc(2) example(1)
modules(0) object-identifiers(0)} -- Remote Operations APDUs          rOSE          ---- FROM Remote-Operations-
APDUs {joint-iso-ccitt          remote-operations(4) apdus(1)} -- Association Control aCSE, aCSE-AS ---- FROM
AssociationControl {joint-iso-ccitt acse(2) .I.bs:Module imports;to be supplied} -- Remote Operations Notation Extension
APPLICATION-CONTEXT, APPLICATION-SERVICE-ELEMENT          ---- FROM Remote-Operations-Notation-extension
{joint-iso-ccitt
remote-operations(4) notation-extension(2)}; -- Application contexts green-use-AC APPLICATION-CONTEXT
    APPLICATION SERVICE ELEMENTS {aCSE} BIND AuthenticateUser UNBIND NoOperation REMOTE
OPERATIONS {rOSE}          INITIATOR CONSUMER OF {green-use-ASE} ABSTRACT SYNTAXES {green-
use-AS, aCSE-AS}          ::= id-ac-g-use green-management-AC APPLICATION-CONTEXT
    APPLICATION SERVICE ELEMENTS {aCSE} BIND AuthenticateManager
UNBIND NoOperation          REMOTE OPERATIONS {rOSE}  INITIATOR CONSUMER OF {green-management-
ASE} ABSTRACT SYNTAXES {green-management-AS, aCSE-AS}          ::= id-ac-g-management NoOperation ::=
UNBIND -- Application service elements green-use-ASE APPLICATION-SERVICE-ELEMENT CONSUMER
INVOKES { green-use-operation-1, ...}          ::= id-ase-g-use green-management-ASE
APPLICATION-SERVICE-ELEMENT          CONSUMER INVOKES {    green-management-
operation-1, ...}
::= id-ase-g-management green-use-operation-1 Green-use-operation-1 ::= 1 ... green-management-operation-1 Green-
management-operation-1 ::= 50 ... -- Abstract syntaxes green-use-AS OBJECT IDENTIFIER ::= id-as-g-use green-
management-AS OBJECT IDENTIFIER ::= id-as-g-management END -- of GreenSystemRealization

```

Annex B (to Recommendation X.407) Reference Definition of Object Identifiers

This annex is an integral part of this Recommendation.

This annex defines for reference purposes various Object Identifiers cited in the ASN.1 modules of annex C. It uses ASN.1.

With the exception of those assigned in annex A, all Object Identifiers this Recommendation assigns are assigned in this annex.

The annex is definitive for all but those for ASN.1 modules and the subject of application service definition conventions itself.

The definitive assignments for the former occur in the modules themselves; other references to them appear in IMPORT clauses. The latter is fixed.

```
.I.mo:ASDCObjectIdentifiers; {joint-iso-ccitt mhs(6) asdc(2) modules(0) object-identifiers(0)}
DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue -- Exports everything. IMPORTS -- nothing -- ; .I.ty:ID; ::=
OBJECT IDENTIFIER -- Abstract Service Definition Conventions (not definitive) .I.va:id-asdc; ID ::= {joint-iso-ccitt mhs(6)
asdc(2)} -- not definitive -- Categories .I.va:id-mod; ID ::= {id-asdc 0} -- modules; not definitive .I.va:id-ex; ID ::= {id-asdc
1} -- example; not definitive -- Modules .I.va:id-mod-object-identifiers; ID ::= {id-mod 0} -- not definitive .I.va:id-mod-
notation; ID ::= {id-mod 1} -- not definitive END -- of ASDCObjectIdentifiers
```

Annex C (to Recommendation X.407) Reference Definition of Notation

This annex is an integral part of this Recommendation.

This annex, a supplement to section two, defines for reference purposes the notation for specifying abstract models and services. It employs ASN.1.

```
-----
.I.mo:AbstractServiceNotation; {joint-iso-ccitt mhs(6) asdc(2) modules(0) notation(1)}
DEFINITIONS IMPLICIT TAGS ::= BEGIN -- Prologue EXPORTS ABSTRACT-BIND, ABSTRACT-ERROR,
ABSTRACT-OPERATION, ABSTRACT-UNBIND, OBJECT, PORT, REFINE; IMPORTS -- Remote Operations
Notation BIND, ERROR, OPERATION, UNBIND ---- FROM Remote-Operation-Notation {joint-iso-
ccitt remote-operations(4) notation(0)}; -- Object macro .I.ma:OBJECT; MACRO ::= BEGIN TYPE NOTATION ::=
"PORTS" "{" PortList "}" | empty VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER) PortList ::= Port " ,"
PortList | Port Port ::= value (PORT) PortType PortType ::= Symmetric | Asymmetric Symmetric ::= empty
Asymmetric ::= Consumer | Supplier Consumer ::= "[C]" Supplier ::= "[S]" END -- Port macro .I.ma:PORT;
MACRO ::= BEGIN TYPE NOTATION ::= Operations | empty VALUE NOTATION ::= value (VALUE OBJECT
IDENTIFIER) Operations ::= Symmetrical | Asymmetrical Symmetrical ::= "ABSTRACT" "OPERATIONS" "{"
OperationList "}" Asymmetrical ::= OneSided | TwoSided OneSided ::= Consumer | Supplier TwoSided ::=
Consumer Supplier | Supplier Consumer Consumer ::= "CONSUMER" "INVOKES" "{" OperationList "}"
Supplier ::= "SUPPLIER" "INVOKES" "{" OperationList "}" OperationList ::= Operation " ," OperationList | Operation
Operation ::= value (ABSTRACT-OPERATION) END -- Refine macro .I.ma:REFINE; MACRO ::= BEGIN TYPE
NOTATION ::= Object "AS" ComponentList VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
ComponentList ::= Component ComponentList | Component Component ::= ObjectSpec PortSpecList ObjectSpec ::=
Object | Object "RECURRING" PortSpecList ::= PortSpec PortSpecList | PortSpec PortSpec ::= value (PORT) PortType
PortStatus PortType ::= Consumer | Supplier | empty Consumer ::= "[C]" Supplier ::= "[S]" PortStatus ::=
"VISIBLE" | "PAIRED" "WITH" ObjectList ObjectList ::= Object " ," ObjectList | Object Object ::= value (OBJECT)
END -- Abstract bind, unbind, operation, and error macros .I.ma:ABSTRACT-BIND; MACRO ::= BEGIN TYPE
NOTATION ::= Ports Bind VALUE NOTATION ::= value (VALUE BindType) Ports ::= "TO" "{" PortList "}" | empty
PortList ::= Port " ," PortList | Port Port ::= value (PORT) PortSide PortSide ::= Consumer | Supplier | empty
Consumer ::= "[C]" Supplier ::= "[S]" Bind ::= type (BindType) -- must be a BIND type | empty
<BindType ::= BIND> END .I.ma:ABSTRACT-UNBIND; MACRO ::= BEGIN TYPE NOTATION ::= Ports Unbind
VALUE NOTATION ::= value (VALUE UnbindType) Ports ::= "FROM" "{" PortList "}" | empty PortList ::= Port
" ," PortList | Port Port ::= value (PORT) PortSide PortSide ::= Consumer | Supplier | empty Consumer ::= "[C]"
Supplier ::= "[S]" Unbind ::= type (UnbindType) -- must be an UNBIND type | empty <UnbindType ::=
UNBIND> END .I.ma:ABSTRACT-OPERATION; MACRO ::= OPERATION .I.ma:ABSTRACT-ERROR; MACRO ::=
ERROR END -- of AbstractServiceNotation
```

Annex D (to Recommendation X.407) Differences Between CCITT Recommendation and ISO Standard

This annex is not a part of this Recommendation.

This annex lists all but the purely stylistic differences between this Recommendation and the corresponding ISO International Standard.

No differences between the two specifications exist.

CCITT Draft Recommendation X.407 MHS: Abstract Service Definition Conventions (Version 5, November 1987, Gloucester)

-- --

Annex E (to Recommendation X.407) Index

This annex indexes this Recommendation. It gives the number(s) of the page(s) on which each item in each of several

categories is defined. Its coverage of each category is exhaustive.

This annex indexes items (if any) in the following categories:

- a) Abbreviations (ab)
- b) Terms (gt)
- c) Information items (ot)
- d) ASN.1 modules (mo)
- e) ASN.1 macros (ma)
- f) ASN.1 types (ty)
- g) ASN.1 values (va)
- h) Bilateral agreements (ba)
- i) Items for further study (fs)
- j) Items to be supplied (fs)

.Begin Index.

Abbreviations

AC 2

APDU 1

ASE 2

ASN.1 2

OSI 2

ROS 2

Terms

abstract bind operation 8

abstract error 10

abstract model 4

abstract object 4

abstract operation 9

abstract port 5

abstract procedure 7

abstract refinement 6

abstract service 6

abstract service provider 6

abstract service user 6

abstract unbind operation 8

argument 7

asymmetric 5

binding 5

bound 5

consumer 5

error information 8

initiator 8

invocation 7

invoker 7

match 5

model 4

object 4

parameter 10

performance 7

performer 7

port 5

procedure 7

provider 6
real 11
refinement 6
responder 8
result 7
supplier 5
symmetric 5
unbinding 5
user 6
Information Items
None
ASN.1 Modules
AbstractServiceNotation 28
ASDCObjectIdentifiers 27
ASN.1 Macros
ABSTRACT-BIND 8, 29
ABSTRACT-ERROR 10, 29
ABSTRACT-OPERATION 9, 29
ABSTRACT-UNBIND 9, 29
OBJECT 4, 28
PORT 5, 28
REFINE 6, 29
ASN.1 Types
ID 27
ASN.1 Values
id-asdc 27
id-ex 27
id-mod 27
id-mod-notation 27
id-mod-object-identifiers 27
Bilateral Agreements
None
Items for Further Study
None
Items to Be Supplied
Module imports 24, 25
.End Index.
ÇîvÃq lû í h= Q d#^ 4^_]
l
ZÇ
æ
U @ Ç @ Ç @ Ç B B @! Ç! É æ
Γ
±
v që ÿ l
gŁ
e
c S a[@ Ç @ @ Ç @ Ç @ Ç [â yĩ w r ^ mC I hL M c∞ ≥ ^] [@ @ Ç @ Ç @ Ç @ Ç @ Ç @ Ç] ÷ vā
è qÅ É l β gπ Σ bÃ ö]ù ÿ Xf

Recommendation X.407 and ISO 10021-3, Information processing systems. Text Communication-MOTIS-Abstract Service Definition Conventions, were developed in close collaboration and are technically aligned.