

INTERNATIONAL STANDARD

ISO/IEC 9506-2

First edition
1990-10-15

Industrial automation systems — Manufacturing Message Specification —

Part 2: Protocol specification

*Systèmes d'automatisation industrielle — Spécification de messagerie industrielle —
Partie 2: Spécification de protocole*



Reference number
ISO 9506-2 : 1990 (E)
AVA Documents
Document Sharing Hub

Contents

	Page
1 Scope	1
1.1 Specifications	1
1.2 Procedures	1
1.3 Applicability	1
1.4 Conformance	1
2 Normative references	1
3 Definitions	2
3.1 Reference Model definitions	3
3.2 Service Convention definitions	3
3.3 Abstract Syntax Notation definitions	3
3.4 Other definitions	4
4 Abbreviations	8
5 Conventions	9
5.1 Service Conventions	9
5.2 Base of Numeric Values	9
5.3 Notation	9
5.4 Supporting Productions	9
5.5 Pass-through Parameters	9
5.6 Negative Confirmation	10
5.7 Modifiers to a Service Request	11
5.8 Presentation of Errors	11
5.9 Use of Companion Standard Fields	11
5.10 Calling and Called MMS-user	11
5.11 Sending and Receiving MMS-user and MPPM	11
5.12 Requesting and Responding MMS-user	11
5.13 Client and Server of a Service	12
5.14 ASN.1 Definitions	12

© ISO/IEC 1990

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 publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

6	Elements of Protocol Procedure	12
6.1	Descriptive Conventions	12
6.2	Entering and Leaving the MMS Environment	12
6.3	Operating in the MMS Environment	13
6.4	Handling of Error Conditions	18
6.5	The Reject Service and RejectPDU	18
7	MMS PDU	18
7.1	The Confirmed-RequestPDU	19
7.2	The Unconfirmed-PDU	20
7.3	The Confirmed-ResponsePDU	20
7.4	The Confirmed-ErrorPDU	20
7.5	Supporting Productions	21
7.6	Common MMS Types	31
8	Environment And General Management Protocol	34
8.1	Introduction	34
8.2	Initiate	34
8.3	Conclude	37
8.4	Abort	37
8.5	Cancel	37
8.6	Reject	38
9	VMD Support Protocol	39
9.1	Introduction	39
9.2	Status	39
9.3	UnsolicitedStatus	40
9.4	GetNameList	40
9.6	Rename	42
9.7	GetCapabilityList	43
10	Domain Management Protocol	44
10.1	Introduction	44
10.2	InitiateDownloadSequence	44
10.3	DownloadSegment	45
10.4	TerminateDownloadSequence	45
10.5	InitiateUploadSequence	46
10.6	UploadSegment	46
10.7	TerminateUploadSequence	47
10.8	RequestDomainDownload	47
10.9	RequestDomainUpload	48
10.10	LoadDomainContent	48
10.11	StoreDomainContent	49
10.12	DeleteDomain	49
10.13	GetDomainAttributes	50

11	Program Invocation Management Protocol	50
11.1	Introduction	51
11.2	CreateProgramInvocation	51
11.3	DeleteProgramInvocation	52
11.4	Start	52
11.5	Stop	53
11.6	Resume	54
11.7	Reset	54
11.8	Kill	55
11.9	GetProgramInvocationAttributes	55
12	Variable Access Protocol	56
12.1	Conventions	56
12.2	Protocol For Specifying Types	57
12.3	Protocol For Specifying Alternate Access	57
12.4	Protocol for Specifying Data Values	59
12.5	Protocol for Specifying Access To Variables	62
12.6	Read	63
12.7	Write	63
12.8	InformationReport	64
12.9	GetVariableAccessAttributes	64
12.10	DefineNamedVariable	65
12.11	DefineScatteredAccess	65
12.12	GetScatteredAccessAttributes	66
12.13	DeleteVariableAccess	66
12.14	DefineNamedVariableList	67
12.15	GetNamedVariableListAttributes	67
12.16	DeleteNamedVariableList	68
12.17	DefineNamedType	68
12.18	GetNamedTypeAttributes	69
12.19	DeleteNamedType	69
13	Semaphore Management Protocol	70
13.1	Introduction	70
13.2	TakeControl	70
13.3	RelinquishControl	71
13.4	DefineSemaphore	71
13.5	DeleteSemaphore	72
13.6	ReportSemaphoreStatus	72
13.7	ReportPoolSemaphoreStatus	72
13.8	ReportSemaphoreEntryStatus	73
13.9	AttachToSemaphore Modifier	74
13.10	Semaphore and Resource Management Supporting Productions	74

14	Operator Communication Protocol	74
14.1	Introduction	74
14.2	Input	74
14.3	Output	75
15	Event Management Protocol	75
15.1	Introduction	75
15.2	DefineEventCondition	76
15.3	DeleteEventCondition	76
15.4	GetEventConditionAttributes	77
15.5	ReportEventConditionStatus	78
15.6	AlterEventConditionMonitoring	79
15.7	TriggerEvent	79
15.8	DefineEventAction	79
15.9	DeleteEventAction	80
15.10	GetEventActionAttributes	81
15.11	ReportEventActionStatus	81
15.12	DefineEventEnrollment	82
15.13	DeleteEventEnrollment	83
15.14	GetEventEnrollmentAttributes	83
15.15	ReportEventEnrollmentStatus	85
15.16	AlterEventEnrollment	86
15.17	EventNotification	86
15.18	AcknowledgeEventNotification	88
15.19	GetAlarmSummary	88
15.20	GetAlarmEnrollmentSummary	89
15.21	AttachToEventCondition	91
15.22	Supporting Productions	91
16	Journal Management Protocol	92
16.1	Introduction	93
16.2	ReadJournal	93
16.3	WriteJournal	94
16.4	InitializeJournal	94
16.5	ReportJournalStatus	95
16.6	CreateJournal	95
16.7	DeleteJournal	95
16.8	Supporting Productions	96
17	Mapping to ACSE and Presentation Services	97
17.1	Mapping of PDUs	97
17.2	Directly-Mapped Abort Service	97
17.3	Construction of MMS PDUs	98

17.4 Delivery of Service Primitives to an MMS-user 98

17.5 Right to Send Data..... 98

17.6 Reliable Underlying Service 98

17.7 Flow Control 98

17.8 Use of Presentation Contexts 98

17.9 Negotiation of MMS Abstract Syntaxes 99

17.10 Termination of Application Association 101

17.11 Abstract Syntax Definition..... 101

17.12 Application Context Name..... 101

18 Conformance..... 102

18.1 Introduction 102

18.2 PICS Part One: Implementation Information..... 102

18.3 PICS Part Two: Service CBBs 103

18.4 PICS Part Three: Parameter CBBs 107

18.5 PICS Part Four: Local Implementation Values..... 107

19 MMS Abstract Syntax 109

Annexes

A Requirements for Companion Standards (normative) 115

B File Access Protocol (normative) 119

C File Management Protocol (informative)..... 121

Index..... 127

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

International Standard ISO/IEC 9506-2 was prepared by Technical Committee ISO/IEC TC 184, *Industrial automation systems*, Sub-Committee SC 5, *System integration and communication*.

ISO/IEC 9506 consists of the following parts, under the general title *Industrial automation systems — Manufacturing Message Specification*:

- *Part 1: Service definition*
- *Part 2: Protocol specification.*

Introduction

This part of ISO/IEC 9506 provides a wide variety of services useful for various manufacturing and process control devices. It is designed to be used both by itself and in conjunction with Companion Standards, which describe the application of subsets of these services to particular device types.

The services provided by the Manufacturing Message Specification (MMS) range from simple to highly complex. It is not expected that all of these services will be supported by all devices. The subset to be supported is limited in some cases by Companion Standards, and in all cases may be limited by the implementor. Characteristics important in selection of a subset of services to be supported include:

- a) applicability of the service to the device;
- b) the complexity of services and requirements;
- c) the complexity of provision of a particular class of service via the network versus the complexity of the device.

Security considerations

When implementing MMS in secure or safety critical applications, features of the OSI security architecture may need to be implemented. Appropriate features should be selected from ISO 7498-2 covering safety architectures and features. Those of particular interest cover the position (in OSI) of

- a) access control;
- b) authentication;
- c) non-repudiation.

Specific implementation methods shall be at the discretion of the implementor.

Complexity of services and requirements

Some MMS services are quite complex and should be considered as advanced functions. Devices used in very simple applications often will not require such advanced functions, and hence will not support such MMS services.

Keywords

Application Interworking
Application Layer Protocol
Information Processing Systems
Manufacturing Communications Network
Manufacturing Message Specification
Numerical Control System
Open Systems Interconnection
OSI Reference Model
Process Control System
Programmable Controller
Programmable Device
Robotics Control System
Virtual Manufacturing Device

General

This part of ISO/IEC 9506 is one of a set of International Standards produced to facilitate the interconnection of information processing systems. It is positioned within the application layer of the Open Systems Interconnection Environment as an Application Service Element (ASE) with respect to other related standards by the Basic Reference Model for Open Systems Interconnection (ISO 7498).

The aim of Open Systems Interconnection is to allow, with a minimum of technical agreement outside the interconnection standards, the interconnection of information processing systems:

- a) from different manufacturers;
- b) under different managements;
- c) of different levels of complexity;
- d) of different ages.

Purpose

The purpose of this part of ISO/IEC 9506 is to define the Manufacturing Message Specification Protocol. It is most closely related to and lies within the field of application of the Manufacturing Message Specification Service Definition, ISO/IEC 9506-1. It also uses and references the Association Control Service Element Definition (ISO 8649) and the Presentation Layer Service Definition (ISO 8822), whose provisions it assumes in order to accomplish the aims of the Manufacturing Message Specification Protocol. The inter-relationship of these International Standards is depicted in Figure 1.

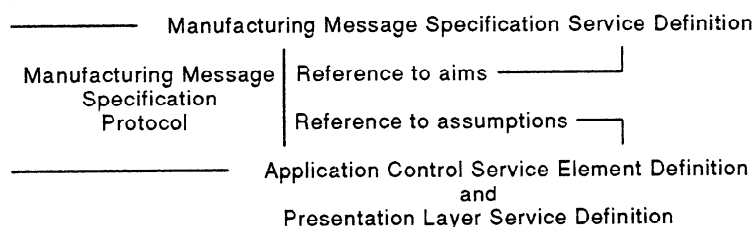


Figure 1 — Relationship between the Manufacturing Message Specification Protocol and adjacent services

The MMS protocol is structured so that subsets of protocol can be defined. The variations and options available within this part of ISO/IEC 9506 are essential to enable a Manufacturing Message Specification to be provided for a wide variety of applications. Thus, a minimally conforming implementation will not be suitable for use in all possible circumstances. It is important, therefore, to qualify all references to this part of ISO/IEC 9506 with statements of the options provided or required with statements of the intended purpose of provision or use.

NOTE — The services of this part of ISO/IEC 9506 are generic, and intended to be referenced by Companion Standards, each of which is directed to a more specific class of application. The services of this part of ISO/IEC 9506 may also be used in a stand-alone manner (without the use of Companion Standards).

It should be noted that, as the number of valid protocol sequences is very large, it is not possible with current technology to verify that an implementation will operate the protocol defined in this part of ISO/IEC 9506 correctly under all circumstances. It is possible by means of testing to establish confidence that an implementation correctly operates the protocol in a representative sample of circumstances. It is, however, intended that this part of ISO/IEC 9506 can be used in circumstances where two implementations fail to communicate in order to determine whether one or both have failed to operate the protocol correctly.

Intended users

The primary aims of this part of ISO/IEC 9506 is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer MMS entities at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- a) as a guide for implementors and designers;
- b) for use in the testing and procurement of equipment;
- c) as part of an agreement for the admittance of systems into the open systems environment;
- d) as a refinement to the understanding of OSI.

This part of ISO/IEC 9506 is concerned, in particular, with the communication and interworking of programmable devices on the plant floor. By using this part of ISO/IEC 9506 together with other standards positioned within the OSI Reference Model, otherwise incompatible systems may work together in any combination.

Industrial automation systems - Manufacturing Message Specification - Part 2: Protocol specification

1 Scope

The Manufacturing Message Specification is an application layer standard designed to support messaging communications to and from programmable devices in a Computer Integrated Manufacturing (CIM) environment.

1.1 Specifications

This part of ISO/IEC 9506 specifies:

- a) procedures for a single protocol for the transfer of data and control information from one application entity to a peer application entity in the MMS-context;
- b) the means of selecting the services to be used by the application entities while communicating in the MMS-context;
- c) the structure of the Manufacturing Messaging Specification Protocol Data Units used for the transfer of data and control information.

1.2 Procedures

The procedures are defined in terms of

- a) the interactions between peer application entities through the exchange of Manufacturing Message Specification Application Protocol Data Units;
- b) the interactions between an MMS-provider and the MMS-user in the same system through the exchange of MMS primitives;
- c) the interactions between an MMS-provider and the Association Control Service Element through the exchange of association control service primitives;
- d) the interactions between an MMS-provider and a presentation service provider through the exchange of Presentation service primitives.

1.3 Applicability

These procedures are applicable to instances of communication between systems which support MMS within the application layer of the OSI Reference Model, and which require the ability to interconnect in an open systems interconnection environment.

1.4 Conformance

This part of ISO/IEC 9506 also specifies conformance requirements for systems implementing these procedures. This part of ISO/IEC 9506 does not contain tests to demonstrate compliance with such requirements.

2 Normative References

ISO/IEC 9506-2: 1990(E)

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9506. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9506 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 646: 1983, *Information processing — ISO 7-bit coded character set for information interchange*.

ISO 7498: 1984, *Information processing systems — Open Systems Interconnection — Basic Reference Model*.

ISO 7498-2: 1989, *Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture*.

ISO 7498-3: 1989, *Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 3: Naming and addressing*.

ISO 8326: 1987, *Information processing systems — Open Systems Interconnection — Basic connection oriented session service definition*.

ISO/TR 8509: 1987, *Information processing systems — Open Systems Interconnection — Service conventions*.

ISO 8571: 1988, *Information processing systems — Open Systems Interconnection — File Transfer, Access and Management*.

ISO 8649: 1988, *Information processing systems — Open Systems Interconnection — Service definition for the Association Control Service Element*.

ISO 8650: 1988, *Information processing systems — Open Systems Interconnection — Protocol specification for the Association Control Service Element*.

ISO 8822: 1988, *Information processing systems — Open Systems Interconnection — Connection oriented presentation service definition*.

ISO 8824: 1987, *Information processing systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1)*.

ISO 8824/Add 1: — ¹⁾, *Information processing systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1) Addendum 1: ASN.1 Extensions*.

ISO 8825: 1987, *Information processing systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.

ISO 8825/Add 1: — ¹⁾, *Information processing systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) Addendum 1: ASN.1 Extensions*.

ISO 9040: — ¹⁾, *Information processing systems — Open Systems Interconnection — Virtual terminal service — Basic class*.

ISO 9041: — ¹⁾, *Information processing systems — Open Systems Interconnection — Virtual terminal protocol — Basic class*.

ISO/IEC 9506-1: 1990, *Industrial automation systems — Manufacturing Message Specification — Part 1: Service definition*.

ISO/IEC 9545-1: 1989, *Information technology — Open Systems Interconnection — Application Layer Structure*.

ISO/IEC 9594: — ¹⁾, *Information processing systems — Open Systems Interconnection — The Directory*.

IEEE 754: 1985, *IEEE Standard for Binary Floating-Point Arithmetic*.

3 Definitions

NOTE — The definitions contained in this clause make use of abbreviations defined in clause 4.

For the purposes of this part of ISO/IEC 9506, the following definitions apply.

¹⁾ To be published.

3.1 Reference Model definitions

This part of ISO/IEC 9506 is based on the concepts developed in the Basic Reference Model for Open Systems Interconnection (ISO 7498), and makes use of the following terms defined in that International Standard:

- a) application-entity;
- b) application-process;
- c) application service element;
- d) open system;
- e) (N)-protocol;
- f) (N)-protocol-data-unit;
- g) (N)-service-access-point;
- h) (N)-layer;
- i) system;
- j) (N)-user-data.

3.2 Service Convention definitions

This part of ISO/IEC 9506 makes use of the following terms defined in the OSI Service Conventions (ISO TR 8509) as they apply to the Manufacturing Message Specification:

- a) confirm;
- b) indication;
- c) primitive;
- d) request;
- e) response;
- f) service primitive;
- g) service provider;
- h) service user.

3.3 Abstract Syntax Notation definitions

This part of ISO/IEC 9506 makes use of the following terms defined in the Abstract Syntax Notation One (ASN.1) Specification (ISO 8824):

- 1) value;
- 2) type;
- 3) simple type;
- 4) structure type;
- 5) component type;
- 6) tag;
- 7) tagging;
- 8) type (or value) reference name;

ISO/IEC 9506-2: 1990(E)

- 9) character string type;
- 10) boolean type;
- 11) true;
- 12) false;
- 13) integer type;
- 14) bitstring type;
- 15) octetstring type;
- 16) null type;
- 17) sequence type;
- 18) sequence-of type;
- 19) tagged type;
- 20) choice type;
- 21) selection type;
- 22) real type;
- 23) object identifier type;
- 24) MACRO;
- 25) module;
- 26) production;
- 27) ASN.1 encoding rules;
- 28) ASN.1 character set;
- 29) external type.

3.4 Other definitions

For the purpose of this part of ISO/IEC 9506, the following definitions also apply:

3.4.1 AA-specific (Application Association specific):

An adjective used to describe an object whose name has a scope that is a single application association (i.e. the name may be referenced only on the application association over which the object was defined).

3.4.2 attribute:

A data element, having a defined meaning, together with a statement of the set of possible values it may take.

3.4.3 Called MMS-user:

The MMS-user that issues the Initiate.response service primitive.

3.4.4 Calling MMS-user:

The MMS-user that issues the Initiate.request service primitive.

3.4.5 Client:

The peer communicating entity which makes use of a VMD for some particular purpose via a service request instance.

3.4.6 conformance building block (CBB):

An atomic unit used to describe MMS conformance requirements.

3.4.7 data:

Any representation to which meaning is or might be assigned (e.g. characters).

3.4.8 domain:

An abstract object that represents a subset of the capabilities of a VMD which is used for a specific purpose.

3.4.9 Domain-specific:

An adjective used to describe an object whose name has a scope that is a single domain (i.e. the name can be referenced over all application associations established with the VMD that may reference this domain).

3.4.10 download:

The process of transferring the content of a domain, including any subordinate objects, via load data to an MMS-user.

3.4.11 event management:

The management of event conditions, event actions, and event enrollments.

3.4.12 file:

An unambiguously named collection of information having a common set of attributes.

3.4.13 file operation:

The transfer of files between open systems, the inspection, modification, or replacement of part of a file's content, or the management of a file and its attributes.

3.4.14 filestore:

An organized collection of files, including their attributes and names, residing at a particular open system.

3.4.15 Information:

The combination of data and the meaning that it conveys.

ISO/IEC 9506-2: 1990(E)

3.4.16 invalid PDU:

A PDU which does not comply with the requirements of this part of ISO/IEC 9506 with respect to structure, semantic meaning, or both.

3.4.17 journal:

A set of recorded, time-tagged event transitions, variable data, and/or comments, which may be logically ordered during retrieval.

3.4.18 local matter:

A decision made by a system concerning its behaviour in the Manufacturing Message Specification that is not subject to the requirements of this part of ISO/IEC 9506.

3.4.19 Manufacturing Message Protocol Machine (MPPM):

An abstract machine that carries out the procedures specified in this part of this part of ISO/IEC 9506.

3.4.20 MMS-context:

A specification of the service elements of MMS and semantics of communication to be used during the lifetime of an application association.

3.4.21 MMS-provider:

That part of the application entity that conceptually provides the MMS service through the exchange of MMS PDUs.

3.4.22 MMS-user:

That portion of the application process which conceptually invokes the Manufacturing Message Specification.

3.4.23 monitored event:

A detected change in the state of an event condition.

3.4.24 network-triggered event:

A trigger which occurs due to an explicit solicitation by a client.

3.4.25 operator station:

An abstract object representing equipment associated with a VMD that provides for input/output interaction with an operator.

3.4.26 predefined object:

An object, whose name is of VMD-specific or Domain-specific scope, that is instantiated through the use of some mechanism other than an MMS service.

3.4.27 program invocation:

An abstract object representing a dynamic element which most closely corresponds to an execution thread in a multi-tasking environment, which is composed of a set of domains.

3.4.28 protocol error:

A PDU that does not comply with the requirements of this part of ISO/IEC 9506.

3.4.29 Receiving MMPM:

The MMPM that receives an MMS PDU.

3.4.30 Receiving MMS-user:

The MMS-user that receives an indication or confirmation service primitive.

3.4.31 remote device control and monitoring:

The manipulation or inspection of the state of a device attached to the responder of a service request.

3.4.32 Requesting MMS-user:

The MMS-user that issues the request service primitive for a service.

3.4.33 Responding MMS-user:

The MMS-user that issues the response service primitive for a service.

3.4.34 semaphore:

A conceptual lock associated with a logical or physical resource that permits access to that resource only by an owner of the lock.

3.4.35 semaphore management:

The control of semaphores.

3.4.36 Sending MMPM:

The MMPM that sends an MMS PDU.

3.4.37 Sending MMS-user:

The MMS-user that issues a request or response service primitive.

3.4.38 Server:

The peer communicating entity which behaves as a VMD for a particular service request instance.

ISO/IEC 9506-2: 1990(E)

3.4.39 standardized object:

An object instantiation, whose name is of VMD-specific or Domain-specific scope, whose definition is provided in this part of ISO/IEC 9506 or an MMS Companion Standard.

3.4.40 type:

An abstract description of the class of data which may be conveyed by the value of a variable.

3.4.41 upload:

The process of transferring the content of a domain, including any subordinate objects, via load data from a remote user, in such a manner as to allow subsequent download.

3.4.42 valid PDU:

A PDU which complies with the requirements of this part of ISO/IEC 9506 with respect to structure and semantic meaning.

3.4.43 variable:

One or more data elements that are referred to together by a single name or description.

3.4.44 variable access:

The inspection or modification of variables or components of variables defined at a VMD.

3.4.45 Virtual Manufacturing Device (VMD):

An abstract representation of a specific set of resources and functionality at a real manufacturing device and a mapping of this abstract representation to the physical and functional aspects of the real manufacturing device.

3.4.46 VMD-specific:

An adjective used to describe an object whose name has a scope that is a single VMD (i.e. the name may be referenced by all application associations established with the VMD).

4 Abbreviations

AA	: application association
ACSE	: Association Control Service Element
AE	: application entity
AP	: application process
APDU	: application protocol data unit
ASE	: application service element
ASN.1	: Abstract Syntax Notation One
CBB	: conformance building block
FRSM	: file read state machine

FTAM	: File Transfer, Access and Management
MMPM	: Manufacturing Message Protocol Machine
MMS	: Manufacturing Message Specification
NC	: Numerical Control
OCS	: operator communication services
OSI	: Open Systems Interconnection
PC	: Programmable Controller
PDU	: protocol data unit
PSAP	: presentation service access point
SAP	: service access point
SDU	: service data unit
ULSM	: upload state machine
VMD	: Virtual Manufacturing Device
VT	: Virtual Terminal

5 Conventions

5.1 Service Conventions

This part of ISO 9506 uses the descriptive conventions contained in the OSI Service Conventions (ISO TR 8509). The model defines the interactions between the MMS-user and the MMS-provider. Information is passed between an MMS-user and an MMS-provider by service primitives, which may convey parameters.

5.2 Base of Numeric Values

This part of ISO/IEC 9506 uses a decimal representation for all numeric values unless otherwise noted.

5.3 Notation

This part of ISO/IEC 9506 uses the abstract syntax notation defined in ISO 8824 (ASN.1 Specification). In keeping with the intent and requirements of the ASN.1 Standard, all type reference symbols begin with an upper case letter. All value references begin with a lower case letter.

5.4 Supporting Productions

Supporting productions introduced in the various clauses of this part of ISO/IEC 9506 are described where they are referenced if they are used primarily in one place. When supporting productions are referenced multiple times from different places, they are collected at the end of the most relevant clause. In any case, an index of productions with page numbers may be found at the end of this part of ISO/IEC 9506.

5.5 Pass-through Parameters

Many of the parameters of the various MMS services are passed from the request primitive via the service's request PDU to the indication primitive or from the response primitive via the service's response PDU to the confirm primitive, without other action being taken by the MMS-provider relative to the parameter.

5.5.1 Pass-through Request Parameters

The type identified by the type reference name shall be the parameter of the same name (see 5.5) from the service's request primitive, and shall appear as the parameter of the same name in the service's indication primitive, if issued. The value of the parameter in the request primitive, indication primitive, and the request PDU are semantically equivalent.

If the parameter is optional and it is omitted from the request service primitive, it shall be absent in the request PDU. If an optional parameter is absent in the request PDU, it shall be absent in the indication service primitive.

If a parameter has a default in the request PDU and this default value is provided in the request service primitive, then the parameter may be absent in the request PDU. If a parameter has a default value in the request PDU and this parameter is absent in the request PDU, then the parameter shall specify the default value in the indication service primitive.

5.5.2 Pass-through Response Parameters

The type identified by the type reference name shall be the parameter of the same name (see 5.5) from the service's response primitive, and shall appear as the parameter of the same name in the service's confirm primitive, if issued. The value of the parameter in the response primitive, confirm primitive, and the response PDU shall be semantically equivalent.

If the parameter is optional and it is omitted from the response service primitive, it shall be absent in the response PDU. If an optional parameter is absent in the response PDU, it shall be absent in the confirm service primitive.

If a parameter has a default in the response PDU and this default value is provided in the response service primitive, then the parameter may be absent in the response PDU. If a parameter has a default value in the response PDU and this parameter is absent in the response PDU, then the parameter shall specify the default value in the confirm service primitive.

5.5.3 Enumerated Values in Parameters

For those parameters in the service description that have enumerated values, the value specified for the corresponding protocol parameter shall be the value of the same name (see 5.5) from the service primitive containing the parameter. The values conveyed in the service primitive, resulting PDU, and the service primitive that results from receipt of the service primitive shall be semantically equivalent.

NOTE — The correspondence between such values is identified in this part of ISO/IEC 9506 through the use of the same names in the service primitives and protocol. In the service specification, such values are specified in all upper case characters. In the protocol specification, the case of the name is chosen so as to satisfy ASN.1 syntax requirements, with the name in upper case characters following the usage in the protocol in a comment.

5.6 Negative Confirmation

Most confirmed MMS services provide for negative confirmation in the case that an error occurs in the processing of the service request by the responding MMS-user. Such negative confirmation shall be indicated by a Result(-) parameter and an "ErrorType" parameter in the service's response primitive. A Result(-) parameter and an "ErrorType" parameter that is semantically equivalent to those parameters in the response primitive shall appear in the confirm service primitive.

The abstract syntax for a negative confirmation shall be the ErrorPDU of the service, with the "error" field derived from the "Problem" parameter in the response service primitive.

5.7 Modifiers to a Service Request

MMS services allow modifiers to be used with instances of service requests.

In instances of requests of services which make use of modifiers, the modifiers specified in a RequestPDU shall be semantically equivalent to, and in the same order as, those modifiers specified in the request service primitive. The indication primitive shall contain a list of modifiers that is semantically equivalent to, and in the same order as, the modifiers in the RequestPDU.

5.8 Presentation of Errors

For each service presented in the body of this part of ISO/IEC 9506, the errors that may result from the use of that service are not presented with the protocol for the service. Errors are specified in a separate clause.

5.9 Use of Companion Standard Fields

In a number of places in the MMS protocol, a parameter is specified as for use by Companion Standards. This allows the definition of the content of this field to be specified via some means outside the scope of this part of ISO/IEC 9506. Use of such fields is specified in clause 19.

5.10 Calling and Called MMS-user

This part of ISO/IEC 9506 makes use of the terms Calling and Called MMS-user. The Calling MMS-user is the MMS-user that issues the Initiate.request service primitive. The Called MMS-user is the MMS-user that issues the Initiate.response service primitive.

NOTE — The use of the term "called" in MMS is not the same as the general usage of the term in OSI. The MMS usage of the term "called" corresponds to the OSI usage of the term "responding". This distinction has been introduced in order to avoid confusion with the Requesting/Responding MMS-user definition given below.

5.11 Sending and Receiving MMS-user and MMPM

This part of ISO/IEC 9506 makes use of the terms Sending and Receiving MMS-user. The Sending MMS-user is the MMS-user that issues a request or response service primitive. The Receiving MMS-user is the MMS-user that receives an indication or confirmation service primitive.

NOTE — It is important to note that, in the course of completion of a confirmed MMS service, both MMS-users will be senders and receivers at one time. The first MMS-user sends the request and receives the confirmation, while the second MMS-user receives the indication and sends the response.

This part of ISO/IEC 9506 makes use of the terms Sending and Receiving MMPM. The Sending MMPM is the MMPM that sends an MMS PDU. The Receiving MMPM is the MMPM that receives an MMS PDU.

5.12 Requesting and Responding MMS-user

This part of ISO/IEC 9506 makes use of the terms Requesting and Responding MMS-user. The Requesting MMS-user is the MMS-user that issues the request service primitive for a service, while the Responding MMS-user is the MMS-user that issues the response service primitive for a service.

NOTE — It is important to note that the use of the term Responding MMS-user differs from the use of the term Responding entity in ACSE and other standards. In those standards, the term is used to reference the entity that responds to a connection request.

5.13 Client and Server of a Service

This part of ISO/IEC 9506 makes use of the terms Client and Server in order to describe the model of the MMS VMD. The Server is defined as the peer communicating entity which behaves as a VMD for a particular service request instance. The Client is the peer communicating entity which makes use of the VMD for some particular purpose via a service request instance. The VMD model is primarily useful in describing the actions of the Server, and thus in describing the commands and responses that a Client may use. A real end system may adopt the Client role, or the Server role, or both during the lifetime of an application association. Use of MMS in the OSI environment is further described in clause 6.

5.14 ASN.1 Definitions

All ASN.1 definitions provided in this part of ISO/IEC 9506, clauses 7 to 16, and annexes B and C, are part of the ASN.1 Module "MMS-General-Module-1". All ASN.1 definitions provided in this part of ISO/IEC 9506, clause 19, are part of the ASN.1 Module "ISO-9506-MMS-1". The beginning and closing statements indicating that each ASN.1 definition provided is a part of this module is omitted in order to make reading of the document easier. Each ASN.1 definition provided implicitly contains the statement:

```
<ModuleName> DEFINITIONS ::= BEGIN
```

at the beginning of the definition and contains the keyword "END" at the end of the definition, where <ModuleName> is the name of the ASN.1 Module of which the definition forms a part.

NOTE – ISO-9506-MMS-1 represents major revision number 1 of the MMS core abstract syntax provided by this part of this part of ISO/IEC 9506.

MMS-General-Module-1 represents major revision number 1 of the MMS general definitions, which are referenced by the Module ISO-9506-MMS-1 and all MMS Companion Standards.

6 Elements of Protocol Procedure

This clause describes the elements of protocol procedure related to the sending and receiving of MMS PDUs and their relation to service primitive events at the MMS-user to MMS-provider boundary.

6.1 Descriptive Conventions

The figures in this clause use a standard state diagram descriptive mechanism. The following text summarizes this mechanism. Note that all state diagrams are shown from the viewpoint of the MMS-provider.

Each state is represented by a box. The name of the state is shown inside the box. Each arrow represents a transition into or out of a state. The head of the arrow points to the output state, which is the state entered as a result of the transition.

Each transition is labeled with the input action that causes the transition, and the output actions to be taken upon the transition. The inputs are shown above the outputs, and are separated from the outputs by a solid horizontal line.

Service primitives to which a "+" is appended indicate a service primitive containing a Result(+) parameter. Service primitives to which a "-" is appended indicate a service primitive containing a Result(-) parameter.

6.2 Entering and Leaving the MMS Environment

The initiate, conclude, and abort services provide the mechanisms for entering and leaving the MMS environment. The model for these services (which describes allowable sequences of events) is described in ISO/IEC 9506-1, clause 8.

6.3 Operating in the MMS Environment

Once in the MMS environment, there may be a number of services outstanding at any instant in time. ISO/IEC 9506 describes the state diagram for each such service request instance independently.

NOTE – Other clauses of this part of ISO/IEC 9506 define additional limitations on allowable sequences of service primitives, and may further restrict the MMS-user.

6.3.1 Confirmed MMS Services

This clause describes the state transitions for all confirmed services that may be invoked within the MMS environment. This set of services consists of all services that are requested through the use of the Confirmed-RequestPDU.

The state transition diagrams in Figure 2 and Figure 3 are applicable for each of the above services, and are applied separately to each instance of each service request. Multiple concurrent service request instances may be outstanding at any given instant in time, subject to sequencing rules stated in other clauses of this Standard.

All PDUs associated with the execution of a single MMS confirmed service instance (these PDUs are the Confirmed-RequestPDU, the Confirmed-ResponsePDU, the Confirmed-ErrorPDU, the Cancel-RequestPDU, the Cancel-ResponsePDU, the Cancel-ErrorPDU, and the RejectPDU) shall be sent in the same presentation context.

6.3.1.1 The Service Requester

NOTE 1 – The order of receipt of the Cancel-ResponsePDU and Confirmed-ErrorPDU(x) in transition 5 of Figure 2 shall not be relevant.

Figure 2 depicts the progression of a confirmed MMS service request from the service requester's point of view. Before the service request primitive is issued, the service is considered to be in the "Requester Idle" state. Upon receipt of a request primitive for any of the MMS confirmed services, the MMS-provider sends a Confirmed-RequestPDU (specifying the invoke ID that unambiguously identifies the service request instance on the application association) and enters the state "Service Pending Requester".

Upon receipt of a Confirmed-ResponsePDU specifying the service previously requested and the invoke ID that specifies the service instance, the MMS-provider issues a confirmation service primitive (specifying the service type and the invoke ID previously requested) to the MMS-user containing a Result(+) parameter. A state transition into the "Requester Idle" state then occurs.

Upon receipt of a Confirmed-ErrorPDU specifying the previously requested service and the invoke ID that specifies the service instance, the MMS-provider issues a confirmation service primitive (specifying the service type and the invoke ID previously requested) to the MMS-user containing a Result(-) parameter. A state transition into the "Requester Idle" state then occurs.

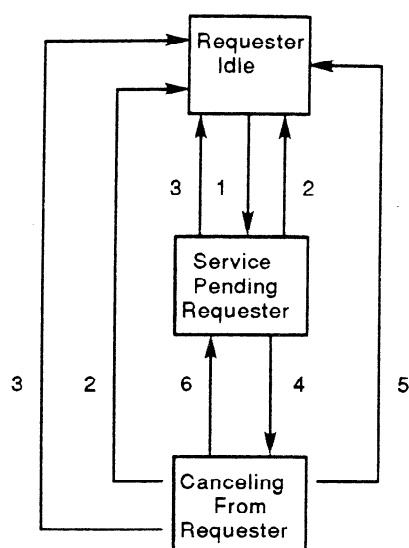
Upon receipt of a cancel request service primitive from the MMS-user, the MMS-provider sends a Cancel-RequestPDU containing the invoke ID of the service request to be cancelled (this information is obtained from the cancel request primitive parameters). The state "Canceling From Requester" is then entered.

The state "Canceling From Requester" is exited upon receipt of any one of four possible input actions. These are described below.

If a Cancel-ErrorPDU is received which specifies an invoke ID that matches the proper instance of the cancel service request, the MMS-provider issues a cancel confirm service primitive to the MMS-user containing a Result(-) parameter and returns to the "Service Pending Requester" state. In this case, the cancel request is considered to have failed.

In the case of a successful cancel request, the following events occur:

- a) a Cancel-ResponsePDU is received whose invoke ID matches the proper instance of the cancel service request;



Transitions:

- | | |
|---|-------------------------------------|
| 1 - <u>x.request</u> | 2 - <u>Confirmed-ResponsePDU(x)</u> |
| Confirmed-RequestPDU(x) | x.confirm+ |
| 3 - <u>Confirmed-ErrorPDU(x)</u> | 4 - <u>cancel.request</u> |
| x.confirm- | Cancel-RequestPDU |
| 5 - <u>Cancel-ResponsePDU and Confirmed-ErrorPDU(x)</u> | |
| cancel.confirm+ and x.confirm- | |
| 6 - <u>Cancel-ErrorPDU</u> | |
| cancel.confirm- | |

Figure 2 – Confirmed Service Request as seen by the Service Requester

- b) a Confirmed-ErrorPDU is received which specifies the service type of the service being cancelled and the invoke ID matches that of the service being cancelled;
- c) the MMS-provider issues a cancel confirm service primitive to the MMS-user containing the Result(+) parameter and a confirm service primitive for the service being cancelled containing a Result(-) parameter (and specifying the cause as cancellation);
- d) the MMS-provider transitions to the "Requester Idle" state.

If a Confirmed-ResponsePDU is received which specifies the service type of the service being cancelled and the invoke ID matches that of the service being cancelled, the MMS-provider issues a confirm service primitive containing a Result(+) parameter for the service that was in the process of being cancelled. In this case, the cancel request is considered to have failed, and a Cancel-ErrorPDU will be received for the cancel service invocation.

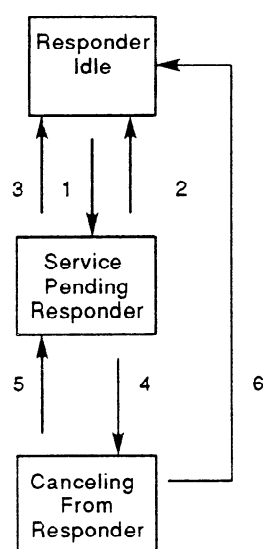
NOTE 2 – This case generally occurs when the Confirmed-ResponsePDU for the service being cancelled and the cancel-RequestPDU are issued simultaneously by the two MMS-users in a two-way simultaneous dialogue.

If a Confirmed-ErrorPDU is received which specifies the service type of the service being cancelled whose invoke ID matches that of the service being cancelled, and the cause for the error is not error class SERVICE-PREEMPT and error code CANCEL, the MMS-provider issues a confirm service primitive containing the Result(-) parameter for the service that was in the process of being cancelled. In this case, the cancel request is considered to have failed, and a Cancel-ErrorPDU will be received for the cancel service invocation.

NOTE 3 — This case generally occurs when the Confirmed-ErrorPDU for the service being cancelled and the Cancel-RequestPDU are issued simultaneously by the two MMS-users in a two-way simultaneous dialogue.

The handling of erroneous cancels is described in 6.4.

6.3.1.2 The Service Responder



Transitions:

- | | |
|---|--|
| 1 - <u>Confirmed-RequestPDU(x)</u>
x.indication | 2 - <u>x.response+</u>
Confirmed-ResponsePDU(x) |
| 3 - <u>x.response-</u>
Confirmed-ErrorPDU(x) | 4 - <u>Cancel-RequestPDU</u>
Cancel.indication |
| 5 - <u>Cancel.response-</u>
Cancel-ErrorPDU | |
| 6 - <u>Cancel.response+ and x.response-</u>
Cancel-ResponsePDU and Confirmed-ErrorPDU(x) | |

Figure 3 — Confirmed Service Request as seen by the Service Responder

NOTE 1 — The order in which the Cancel.response+ and x.response- service primitives are issued in transition 6 of Figure 3 shall not be relevant.

Figure 3 depicts the progression of a confirmed MMS service request from the service responder's point of view. Before the service Confirmed-RequestPDU is received, the service is considered to be in the "Responder Idle" state. Upon receipt of a Confirmed-RequestPDU for any of the confirmed services identified above, the MMS-provider issues an indication primitive (specifying the particular service being requested and an invoke ID that specifies the service instance) and enters the state "Service Pending Responder".

Upon receipt of a response service primitive containing a Result(+) parameter (specifying the service previously indicated and an invoke ID that specifies the service instance), the MMS-provider sends a Confirmed-ResponsePDU (specifying the service type and the invoke ID from the response primitive). A state transition into the "Responder Idle" state then occurs.

Upon receipt of a response service primitive containing a Result(-) parameter (specifying the service previously indicated and an invoke ID that specifies the service instance), the MMS-provider sends a Confirmed-ErrorPDU (specifying the service type and the invoke ID from the response primitive). A state transition into the "Responder Idle" state then occurs.

Upon receipt of a Cancel-RequestPDU specifying the invoke ID of the matching service instance, the MMS-provider issues a cancel indication service primitive specifying the invoke ID of the service request to be cancelled (this information is obtained from the Cancel-RequestPDU parameters). The state "Canceling Service Responder" is then entered.

NOTE 2 — Actions to be taken upon receipt of a Cancel-RequestPDU whose invoke ID does not match any outstanding service instance are specified in 6.4.

The state "Canceling Service Responder" is exited upon receipt of either one of two possible input actions. These are described in the next two paragraphs.

When a cancel request succeeds at the responding MMS-user, the following sequence of events occurs:

- a) the responding MMS-user issues a cancel response specifying the invoke ID of the matching service instance and containing a Result(+) parameter to the MMS-provider, and issues a response service primitive containing a Result(-) parameter (specifying the error class SERVICE and error code CANCEL) for the service being cancelled;
- b) the MMS-provider sends a Cancel-ResponsePDU and a Confirmed-ErrorPDU specifying the service instance cancelled (with error class SERVICE and error code CANCEL);
- c) the MMS-provider returns to the "Responder Idle" state.

The MMS-user shall not issue a cancel response service primitive containing a Result(+) parameter without also issuing a response service primitive containing a Result(-) parameter that specifies the error class SERVICE-PREEMPT and error code CANCEL. Conversely, the MMS-user shall not issue a response service primitive containing a Result(-) parameter that specifies the error class SERVICE-PREEMPT and error code CANCEL without also issuing a cancel response service primitive containing a Result(+) parameter. Hence, these two events logically occur together.

If a cancel response specifying the invoke ID of the matching service instance containing a Result(-) parameter is received, the MMS-provider sends a Cancel-ErrorPDU and returns to the "service pending" state. In this case, the cancel request is considered to have failed.

NOTE 3 — The handling of erroneous cancel requests and invalid PDUs is described in 6.4.

6.3.2 Unconfirmed MMS Services

This clause describes the operation of unconfirmed MMS services. This set of services is defined as those services that make up the UnconfirmedService choice defined in clause 7.

The state transition diagrams in Figures 4 and 5 are applicable for each of the above identified services, and are applied separately to each instance of each service request.

6.3.2.1 The Service Requester

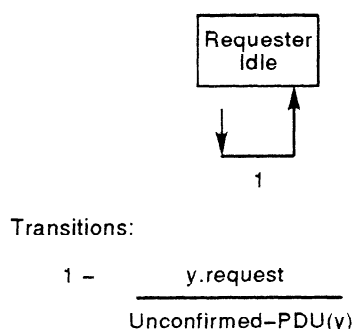


Figure 4 – Unconfirmed Service as seen by the Service Requester

Figure 4 depicts the progression of an unconfirmed MMS service from the service requester's point of view. Before the service request primitive is issued, the service is considered to be in the "Requester Idle" state. Upon receipt of a request primitive for any of the above unconfirmed services, the MMS-provider sends an Unconfirmed-PDU (specifying the particular service being requested) and transitions back to the "Requester Idle" state.

For unconfirmed MMS services, no response PDU or error PDU will be received. Further, it is not possible to cancel an unconfirmed MMS service.

6.3.2.2 The Service Responder

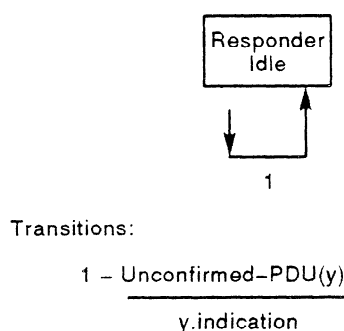


Figure 5 – Unconfirmed Service as seen by the Service Responder

Figure 5 depicts the progression of an unconfirmed MMS service from the service responder's point of view. Before the Unconfirmed-PDU is received, the service is considered to be in the "Responder Idle" state. Upon receipt of a request primitive for any of the above unconfirmed services, the MMS-provider issues an indication service primitive (specifying the particular service being requested based upon information in the Unconfirmed-PDU received) and transitions from the "Responder Idle" state to the "Responder Idle" state (into the same state).

For unconfirmed MMS services, the MMS-user may not issue any response primitive. Further, it is not possible to cancel an unconfirmed MMS service.

6.3.3 The Cancel Service

The Cancel service, while it is a confirmed service, does not operate in the same manner as other confirmed services. When the Cancel service is invoked, no new state machine is created. Rather, the state machine of the service being cancelled is affected. A Cancel service request cannot be cancelled by another invocation of the Cancel service. The invoke ID specified in the Cancel service request may not be that of another invocation of the Cancel service, since the Cancel service only operates on those services that make up the ConfirmedServiceRequest choice defined in clause 7.

At any given instant, only one Cancel service invocation may be outstanding for any given service request instance. An invocation of the Cancel service does not affect the limit on the number of service requests that may be outstanding at any given instant as negotiated by the initiate service. Cancel service requests are not counted in determining if the limit has been reached.

The operation of the Cancel service is described in the previous clauses of this document describing the service requester and service responder for MMS confirmed services.

6.4 Handling of Error Conditions

Upon receipt of an invalid PDU, the MMS-provider shall issue a reject indication service primitive to the MMS-user identifying the error detected, and shall send a RejectPDU to the system from which the invalid PDU was received. In this case, no state change shall occur.

Upon receipt of a Cancel-RequestPDU that attempts to cancel an unknown service request, for example, when the invoke ID specified is not an outstanding confirmed service, the MMS-provider shall send a Cancel-ErrorPDU to the sender of the Cancel request. In this case, the MMS-user is not notified of the erroneous cancel attempt.

NOTE 1 – It is possible for a Cancel-RequestPDU and a Confirmed-ResponsePDU or Confirmed-ErrorPDU for the service requested to be cancelled to be issued concurrently by the two communicating MMS-users. Thus, one side considers the service to have completed, while the other considers it to be awaiting cancellation. In this case, the Cancel request fails and the service completes normally.

Upon receipt of a Cancel-ErrorPDU, where the state machine referred to by the invoke ID of the service to be cancelled is in the "Requester Idle" state, the MMS-provider issues a cancel confirm- service primitive to the MMS-user.

NOTE 2 – This case occurs when a Confirmed-ResponsePDU or Confirmed-ErrorPDU for the service to be cancelled passes a Cancel-RequestPDU for that service.

6.5 The Reject Service and RejectPDU

The reject service is used to notify an MMS-user of a protocol error that has occurred. The operation of this service is described in ISO/IEC 9506-1, clauses 8 and 17.

NOTE – The actions to be taken by an MMS-user upon receipt of a reject indication service primitive are a local matter. It is important to note that, as a result of the possibility that a request, response, or error may be rejected, the two MMS-users involved in a dialogue may not have a common understanding of the state of outstanding transaction objects (see ISO/IEC 9506-1, clause 7). The abort service may be used at any time by an MMS-user to terminate the MMS-environment and the application association.

7 MMS PDU

This clause describes the PDUs used to operate the MMS protocol. The mapping of these PDUs to underlying services is described in clause 17. The mapping of MMS services to these PDUs is described in clauses 8 to 16.

This module defines all the common syntax to MMS and its Companion Standards.

```

MMS-General-Module-1 {iso standard 9506 part(2) mms-general-module-version1(2)}
DEFINITIONS ::= BEGIN

    EXPORTS MMSpdu;
    -- By this statement we imply that all productions subordinate
    -- to MMSpdu are exported also.

    IMPORTS AP-title, AP-invocation-id, AE-qualifier, AE-invocation-id
    FROM ISO-8650-ACSE-1
        {iso standard 8650 abstract-syntax(2) acse-pdi(1)};

    MMSpdu ::= CHOICE {
        confirmed-RequestPDU      [0] IMPLICIT Confirmed-RequestPDU,
        confirmed-ResponsePDU     [1] IMPLICIT Confirmed-ResponsePDU,
        confirmed-ErrorPDU        [2] IMPLICIT Confirmed-ErrorPDU,
        unconfirmed-PDU           [3] IMPLICIT Unconfirmed-PDU,
        rejectPDU                 [4] IMPLICIT RejectPDU,
        cancel-RequestPDU         [5] IMPLICIT Cancel-RequestPDU,
        cancel-ResponsePDU        [6] IMPLICIT Cancel-ResponsePDU,
        cancel-ErrorPDU           [7] IMPLICIT Cancel-ErrorPDU,
        initiate-RequestPDU       [8] IMPLICIT Initiate-RequestPDU,
        initiate-ResponsePDU      [9] IMPLICIT Initiate-ResponsePDU,
        initiate-ErrorPDU         [10] IMPLICIT Initiate-ErrorPDU,
        conclude-RequestPDU       [11] IMPLICIT Conclude-RequestPDU,
        conclude-ResponsePDU      [12] IMPLICIT Conclude-ResponsePDU,
        conclude-ErrorPDU        [13] IMPLICIT Conclude-ErrorPDU
    }

```

There are fourteen types of PDUs in MMS. The Initiate-RequestPDU, the Initiate-ResponsePDU, the Initiate-ErrorPDU, the Conclude-RequestPDU, the Conclude-ResponsePDU, the Conclude-ErrorPDU, the RejectPDU, the Cancel-RequestPDU, the Cancel-ResponsePDU, and the Cancel-ErrorPDU are defined in clause 8. The remaining PDU types are defined in 7.1 to 7.4.

7.1 The Confirmed-RequestPDU

```

Confirmed-RequestPDU ::= SEQUENCE {
    invokeID Unsigned32,
    listOfModifier SEQUENCE OF Modifier OPTIONAL,
    ConfirmedServiceRequest,
    [79] CS-Request-Detail OPTIONAL
    -- shall not be transmitted if value is NULL
}

```

The Confirmed-RequestPDU is a sequence containing four elements, an invokeID, an optional list of modifiers, a ConfirmedServiceRequest, and a CS-Request-Detail.

The invokeID shall be a 32-bit unsigned integer, which shall unambiguously identify a service request among all outstanding confirmed service requests from a particular MMS-user on a given application association. At any instant in time, there shall be at most one service request outstanding from a particular MMS-user on an application association for any given invokeID. The value of the invokeID shall be the value provided by the MMS-user in the request service primitive (see ISO/IEC 9506-1, clause 5). The invokeID provided in the Confirmed-ResponsePDU and Confirmed-ErrorPDU allows the MMS-provider and the MMS-user to correlate these PDUs with the appropriate service request.

The list of modifiers shall be used to prescribe modifiers to the execution of service requests. A modifier specified in a list of modifiers shall be successfully executed before the next modifier in the list of modifiers or before execution of the service request. The order of the modifiers in the list is therefore important. If no list of modifiers is present, then service request execution may begin immediately upon receipt of the request and without pre-conditions.

The ConfirmedServiceRequest shall be used to identify a confirmed service and the argument for that confirmed service. This parameter is further described in 7.5.2.

The CS-Request-Detail shall be an additional parameter to be supplied by a Companion Standard or the abstract syntax defined in clause 19. The identifier of the CHOICE in this parameter shall be identical to the identifier of the CHOICE in the ConfirmedServiceRequest. This parameter is further described in 7.5.2.

7.2 The Unconfirmed-PDU

```
Unconfirmed-PDU ::= SEQUENCE {  
    UnconfirmedService,  
    [79] CS-Unconfirmed-Detail OPTIONAL  
    -- shall not be transmitted if value is NULL  
}
```

The Unconfirmed-PDU shall be a sequence containing an UnconfirmedService and a CS-Unconfirmed-Detail.

The UnconfirmedService shall be used to identify an unconfirmed service and its argument.

The CS-Unconfirmed-Detail shall be an additional parameter to be supplied by a Companion Standard or the abstract syntax defined in clause 19. The identifier of the CHOICE in this parameter shall be identical to the identifier of the CHOICE in the UnconfirmedService. This parameter is further described in 7.5.3.

7.3 The Confirmed-ResponsePDU

```
Confirmed-ResponsePDU ::= SEQUENCE {  
    invokeID Unsigned32,  
    ConfirmedServiceResponse,  
    [79] CS-Response-Detail OPTIONAL  
    -- shall not be transmitted if value is NULL  
}
```

The Confirmed-ResponsePDU shall be a sequence containing three elements, an invokeID, a ConfirmedServiceResponse and a CS-Response-Detail.

The invokeID shall be a 32-bit unsigned integer that shall unambiguously identify a service request among all outstanding confirmed service requests from a particular MMS-user on an application association. At any instant in time, there may be at most one service request outstanding from a particular MMS-user on an application association for any given invokeID. The value of the invokeID shall be the value provided by the MMS-user in the response service primitive (see ISO/IEC 9506-1, clause 5), and shall identify the request instance that caused the service to be carried out. The invokeID in this PDU allows the MMS-provider and MMS-user to correlate this PDU with the appropriate service request.

The ConfirmedServiceResponse shall be used to identify a confirmed service and the response for that confirmed service. This parameter is further described in 7.5.4.

The CS-Response-Detail shall be an additional parameter to be supplied by a Companion Standard or the abstract syntax defined in clause 19. The identifier of the CHOICE in this parameter shall be identical to the identifier of the CHOICE in the ConfirmedServiceResponse. This parameter is further described in 7.5.4.

7.4 The Confirmed-ErrorPDU

```
Confirmed-ErrorPDU ::= SEQUENCE {  
    invokeID [0] IMPLICIT Unsigned32,  
    modifierPosition [1] IMPLICIT Unsigned32 OPTIONAL,  
    serviceError [2] IMPLICIT ServiceError  
}
```

The Confirmed-ErrorPDU shall be a sequence containing three elements, an invokeID, an optional modifierPosition, and a ServiceError.

The `invokeID` shall be a 32-bit unsigned integer that shall unambiguously identify a service request among all outstanding confirmed service requests from a particular MMS-user on an application association. At any instant in time, there may be at most one service request outstanding from a particular MMS-user on an application association for any given `invokeID`. The value of the `invokeID` shall be the value provided by the MMS-user in the response service primitive (see ISO/IEC 9506-1, clause 5), and shall identify the request instance that caused the service to be carried out. The `invokeID` in this PDU allows the MMS-provider and MMS-user to correlate this PDU with the appropriate service request.

The `modifierPosition` shall be a 32-bit unsigned integer that shall unambiguously identify a modifier among all modifiers which were specified in the `listOfModifier` sequence in the `Confirmed-RequestPDU` identified by the `invokeID`. This parameter is derived from the `Modifier Position` sub-parameter in the `Service Error` parameter from the response service primitive (see ISO/IEC 9506-1, clause 17).

The `ServiceError` shall be used to identify the error class and error code for either the modifier of the confirmed service, or the confirmed service. The `ServiceError` parameter is described further in 7.5.5.

7.5 Supporting Productions

The following supporting productions provide definitions used in specifying the MMS PDUs.

7.5.1 Modifier

```
Modifier ::= CHOICE {
    attachToEventCondition [0] IMPLICIT AttachToEventCondition,
    attachToSemaphore      [1] IMPLICIT AttachToSemaphore
}
```

The `Modifier` parameter shall be a choice of one of the MMS modifiers. Each modifier shall serve to modify the normal execution of an MMS confirmed service by placing a pre-condition on the execution of a service request instance which shall be successfully executed before service execution can begin. The effect of a modifier is modeled by the service invocation state machine provided in clause 6. Definitions of modifiers may be found in the protocol descriptions in the following clauses of this part of ISO/IEC 9506.

7.5.2 ConfirmedServiceRequest

```

ConfirmedServiceRequest ::= CHOICE {
    status [0] IMPLICIT Status-Request,
    getNameList [1] IMPLICIT GetNameList-Request,
    identify [2] IMPLICIT Identify-Request,
    rename [3] IMPLICIT Rename-Request,
    read [4] IMPLICIT Read-Request,
    write [5] IMPLICIT Write-Request,
    getVariableAccessAttributes [6] GetVariableAccessAttributes-Request,
    defineNamedVariable [7] IMPLICIT DefineNamedVariable-Request,
    defineScatteredAccess [8] IMPLICIT DefineScatteredAccess-Request,
    getScatteredAccessAttributes [9] GetScatteredAccessAttributes-Request,
    deleteVariableAccess [10] IMPLICIT DeleteVariableAccess-Request,
    defineNamedVariableList [11] IMPLICIT DefineNamedVariableList-Request,
    getNamedVariableListAttributes [12]
        GetNamedVariableListAttributes-Request,
    deleteNamedVariableList [13] IMPLICIT DeleteNamedVariableList-Request,
    defineNamedType [14] IMPLICIT DefineNamedType-Request,
    getNamedTypeAttributes [15]
        GetNamedTypeAttributes-Request,
    deleteNamedType [16] IMPLICIT DeleteNamedType-Request,
    input [17] IMPLICIT Input-Request,
    output [18] IMPLICIT Output-Request,
    takeControl [19] IMPLICIT TakeControl-Request,
    relinquishControl [20] IMPLICIT RelinquishControl-Request,
    defineSemaphore [21] IMPLICIT DefineSemaphore-Request,
    deleteSemaphore [22]
        DeleteSemaphore-Request,
    reportSemaphoreStatus [23]
        ReportSemaphoreStatus-Request,
    reportPoolSemaphoreStatus [24] IMPLICIT
        ReportPoolSemaphoreStatus-Request,
    reportSemaphoreEntryStatus [25] IMPLICIT
        ReportSemaphoreEntryStatus-Request,
    initiateDownloadSequence [26] IMPLICIT
        InitiateDownloadSequence-Request,
    downloadSegment [27] IMPLICIT DownloadSegment-Request,
    terminateDownloadSequence [28] IMPLICIT
        TerminateDownloadSequence-Request,
    initiateUploadSequence [29] IMPLICIT InitiateUploadSequence-Request,
    uploadSegment [30] IMPLICIT UploadSegment-Request,
    terminateUploadSequence [31] IMPLICIT TerminateUploadSequence-Request,
    requestDomainDownload [32] IMPLICIT RequestDomainDownload-Request,
    requestDomainUpload [33] IMPLICIT RequestDomainUpload-Request,
    loadDomainContent [34] IMPLICIT LoadDomainContent-Request,
    storeDomainContent [35] IMPLICIT StoreDomainContent-Request,
    deleteDomain [36] IMPLICIT DeleteDomain-Request,
    getDomainAttributes [37] IMPLICIT GetDomainAttributes-Request,
    createProgramInvocation [38] IMPLICIT CreateProgramInvocation-Request,
    deleteProgramInvocation [39] IMPLICIT DeleteProgramInvocation-Request,
    start [40] IMPLICIT Start-Request,
    stop [41] IMPLICIT Stop-Request,
    resume [42] IMPLICIT Resume-Request,
    reset [43] IMPLICIT Reset-Request,
    kill [44] IMPLICIT Kill-Request,
    getProgramInvocationAttributes [45] IMPLICIT
        GetProgramInvocationAttributes-Request,
    obtainFile [46] IMPLICIT ObtainFile-Request,
    defineEventCondition [47] IMPLICIT DefineEventCondition-Request,
    deleteEventCondition [48]
        DeleteEventCondition-Request,
    getEventConditionAttributes [49]
        GetEventConditionAttributes-Request,
    reportEventConditionStatus [50]
        ReportEventConditionStatus-Request,
    alterEventConditionMonitoring [51] IMPLICIT
        AlterEventConditionMonitoring-Request,
    triggerEvent [52] IMPLICIT TriggerEvent-Request,
    defineEventAction [53] IMPLICIT DefineEventAction-Request,
    deleteEventAction [54]
        DeleteEventAction-Request,

```

```

getEventActionAttributes      [55]      GetEventActionAttributes-Request,
reportEventActionStatus      [56]      ReportEventActionStatus-Request,
defineEventEnrollment        [57]      IMPLICIT DefineEventEnrollment-Request,
deleteEventEnrollment        [58]      DeleteEventEnrollment-Request,
alterEventEnrollment         [59]      IMPLICIT AlterEventEnrollment-Request,
reportEventEnrollmentStatus  [60]      ReportEventEnrollmentStatus-Request,
getEventEnrollmentAttributes [61]      IMPLICIT
                                GetEventEnrollmentAttributes-Request,
acknowledgeEventNotification [62]      IMPLICIT
                                AcknowledgeEventNotification-Request,
getAlarmSummary               [63]      IMPLICIT GetAlarmSummary-Request,
getAlarmEnrollmentSummary    [64]      IMPLICIT
                                GetAlarmEnrollmentSummary-Request,
readJournal                   [65]      IMPLICIT ReadJournal-Request,
writeJournal                  [66]      IMPLICIT WriteJournal-Request,
initializeJournal             [67]      IMPLICIT InitializeJournal-Request,
reportJournalStatus           [68]      ReportJournalStatus-Request,
createJournal                 [69]      IMPLICIT CreateJournal-Request,
deleteJournal                 [70]      IMPLICIT DeleteJournal-Request,
getCapabilityList             [71]      IMPLICIT GetCapabilityList-Request,
fileOpen                      [72]      IMPLICIT FileOpen-Request,
fileRead                      [73]      IMPLICIT FileRead-Request,
fileClose                     [74]      IMPLICIT FileClose-Request,
fileRename                    [75]      IMPLICIT FileRename-Request,
fileDelete                    [76]      IMPLICIT FileDelete-Request,
fileDirectory                 [77]      IMPLICIT FileDirectory-Request,
additionalService             [78]      AdditionalService-Request
    -- choices [72] through [77] are reserved for use by services
    -- defined in annex C
    -- choice [79] is reserved
}

CS-Request-Detail ::= CHOICE {
    status                [0]      IMPLICIT CS-Status-Request,
    getNameList           [1]      IMPLICIT CS-GetNameList-Request,
    input                 [17]     IMPLICIT CS-Input-Request,
    output                [18]     IMPLICIT CS-Output-Request,
    initiateDownloadSequence [26]   IMPLICIT
                                CS-InitiateDownloadSequence-Request,
    downloadSegment       [27]     IMPLICIT CS-DownloadSegment-Request,
    terminateDownloadSequence [28]  IMPLICIT
                                CS-TerminateDownloadSequence-Request,
    initiateUploadSequence [29]     IMPLICIT
                                CS-InitiateUploadSequence-Request,
    uploadSegment         [30]     IMPLICIT CS-UploadSegment-Request,
    terminateUploadSequence [31]    IMPLICIT
                                CS-TerminateUploadSequence-Request,
    requestDomainDownload [32]     IMPLICIT
                                CS-RequestDomainDownload-Request,
    requestDomainUpload   [33]     IMPLICIT CS-RequestDomainUpload-Request,
    loadDomainContent     [34]     IMPLICIT CS-LoadDomainContent-Request,
    storeDomainContent    [35]     IMPLICIT CS-StoreDomainContent-Request,
    deleteDomain          [36]     IMPLICIT CS-DeleteDomain-Request,
    getDomainAttributes   [37]     IMPLICIT CS-GetDomainAttributes-Request,
    createProgramInvocation [38]    IMPLICIT
                                CS-CreateProgramInvocation-Request,
    deleteProgramInvocation [39]   IMPLICIT
                                CS-DeleteProgramInvocation-Request,
    start                 [40]     IMPLICIT CS-Start-Request,
    stop                  [41]     IMPLICIT CS-Stop-Request,
    resume                [42]     IMPLICIT CS-Resume-Request,
    reset                 [43]     IMPLICIT CS-Reset-Request,
    kill                  [44]     IMPLICIT CS-Kill-Request,

```

```

getProgramInvocationAttributes [45] IMPLICIT
                                CS-GetProgramInvocationAttributes-Request,
defineEventCondition          [47] IMPLICIT CS-DefineEventCondition-Request,
deleteEventCondition          [48] IMPLICIT CS-DeleteEventCondition-Request,
getEventConditionAttributes   [49] IMPLICIT
                                CS-GetEventConditionAttributes-Request,
reportEventConditionStatus    [50] IMPLICIT
                                CS-ReportEventConditionStatus-Request,
alterEventConditionMonitoring [51] IMPLICIT
                                CS-AlterEventConditionMonitoring-Request,
triggerEvent                  [52] IMPLICIT CS-TriggerEvent-Request,
defineEventAction             [53] IMPLICIT CS-DefineEventAction-Request,
deleteEventAction             [54] IMPLICIT CS-DeleteEventAction-Request,
getEventActionAttributes      [55] IMPLICIT
                                CS-GetEventActionAttributes-Request,
reportEventActionStatus       [56] IMPLICIT
                                CS-ReportEventActionStatus-Request,
defineEventEnrollment        [57] IMPLICIT
                                CS-DefineEventEnrollment-Request,
deleteEventEnrollment        [58] IMPLICIT
                                CS-DeleteEventEnrollment-Request,
alterEventEnrollment         [59] IMPLICIT CS-AlterEventEnrollment-Request,
reportEventEnrollmentStatus   [60] IMPLICIT
                                CS-ReportEventEnrollmentStatus-Request,
getEventEnrollmentAttributes [61] IMPLICIT
                                CS-GetEventEnrollmentAttributes-Request,
acknowledgeEventNotification [62] IMPLICIT
                                CS-AcknowledgeEventNotification-Request,
getAlarmSummary               [63] IMPLICIT CS-GetAlarmSummary-Request,
getAlarmEnrollmentSummary     [64] IMPLICIT
                                CS-GetAlarmEnrollmentSummary-Request,
readJournal                   [65] IMPLICIT CS-ReadJournal-Request,
writeJournal                   [66] IMPLICIT CS-WriteJournal-Request,
initializeJournal              [67] IMPLICIT CS-InitializeJournal-Request,
reportJournalStatus           [68] IMPLICIT
                                CS-ReportJournalStatus-Request,
createJournal                  [69] IMPLICIT CS-CreateJournal-Request,
deleteJournal                  [70] IMPLICIT CS-DeleteJournal-Request,
getCapabilityList              [71] IMPLICIT CS-GetCapabilityList-Request
}

```

The ConfirmedServiceRequest parameter shall identify the service type and the argument for that service. The context tags provided identify the service type. Definitions for each individual service specify the form of the argument for the service through definition of a type, which is referenced by the ConfirmedServiceRequest production. Each of the services in the ConfirmedServiceRequest choice is a confirmed service.

The AdditionalService-Request shall be reserved for definition of additional services by MMS Companion Standards.

NOTE – All such additional services shall meet the following requirements as described in annex A:

- a) no additional service shall violate any of the state transition rules provided by ISO/IEC 9506-1 and ISO/IEC 9506-2. This limits Companion Standard to definition of substates within existing MMS states;
- b) no additional service shall violate any of the object models provided by ISO/IEC 9506-1;
- c) no additional service shall be used to circumvent the spirit and intent of ISO/IEC 9506-1 and ISO/IEC 9506-2.

7.5.3 UnconfirmedService

```

UnconfirmedService ::= CHOICE {
    informationReport    [0] IMPLICIT InformationReport,
    unsolicitedStatus    [1] IMPLICIT UnsolicitedStatus,
    eventNotification    [2] IMPLICIT EventNotification,
    additionalService    [3] AdditionalUnconfirmedService
}

CS-Unconfirmed-Detail ::= CHOICE {
    unsolicitedStatus    [1] IMPLICIT CS-UnsolicitedStatus,
    eventNotification    [2] IMPLICIT CS-EventNotification
}

```

The UnconfirmedService parameter shall identify the service type and the argument for that service. The context tags provided identify the service type. Definitions for each individual service specify the form of the argument for the service through definition of a type, which is referenced by the UnconfirmedService production. Each of the services in the UnconfirmedService choice is an unconfirmed service.

The AdditionalUnconfirmedService shall be reserved for definition of additional services by MMS Companion Standards.

NOTE – All such additional services shall meet the following requirements as described in annex A:

- a) no additional service shall violate any of the state transition rules provided by ISO/IEC 9506-1 and ISO/IEC 9506-2. This limits Companion Standard to definition of substates within existing MMS states;
- b) no additional service shall violate any of the object models provided by ISO/IEC 9506-1;
- c) no additional service shall be used to circumvent the spirit and intent of ISO/IEC 9506-1 and ISO/IEC 9506-2.

7.5.4 ConfirmedServiceResponse

```

ConfirmedServiceResponse ::= CHOICE {
    status                [0] IMPLICIT Status-Response,
    getNameList           [1] IMPLICIT GetNameList-Response,
    identify              [2] IMPLICIT Identify-Response,
    rename                [3] IMPLICIT Rename-Response,
    read                  [4] IMPLICIT Read-Response,
    write                 [5] IMPLICIT Write-Response,
    getVariableAccessAttributes [6] IMPLICIT
        GetVariableAccessAttributes-Response,
    defineNamedVariable   [7] IMPLICIT DefineNamedVariable-Response,
    defineScatteredAccess [8] IMPLICIT DefineScatteredAccess-Response,
    getScatteredAccessAttributes [9] IMPLICIT
        GetScatteredAccessAttributes-Response,
    deleteVariableAccess [10] IMPLICIT DeleteVariableAccess-Response,
    defineNamedVariableList [11] IMPLICIT
        DefineNamedVariableList-Response,
    getNamedVariableListAttributes [12] IMPLICIT
        GetNamedVariableListAttributes-Response,
    deleteNamedVariableList [13] IMPLICIT
        DeleteNamedVariableList-Response,
    defineNamedType       [14] IMPLICIT DefineNamedType-Response,
    getNamedTypeAttributes [15] IMPLICIT GetNamedTypeAttributes-Response,
    deleteNamedType       [16] IMPLICIT DeleteNamedType-Response,
    input                 [17] IMPLICIT Input-Response,
    output                [18] IMPLICIT Output-Response,
    takeControl            [19] IMPLICIT TakeControl-Response,
    relinquishControl      [20] IMPLICIT RelinquishControl-Response,
    defineSemaphore        [21] IMPLICIT DefineSemaphore-Response,
    deleteSemaphore        [22] IMPLICIT DeleteSemaphore-Response,
    reportSemaphoreStatus  [23] IMPLICIT ReportSemaphoreStatus-Response,
}

```

reportPoolSemaphoreStatus	[24]	IMPLICIT	ReportPoolSemaphoreStatus-Response,
reportSemaphoreEntryStatus	[25]	IMPLICIT	ReportSemaphoreEntryStatus-Response,
initiateDownloadSequence	[26]	IMPLICIT	InitiateDownloadSequence-Response,
downloadSegment	[27]	IMPLICIT	DownloadSegment-Response,
terminateDownloadSequence	[28]	IMPLICIT	TerminateDownloadSequence-Response,
initiateUploadSequence	[29]	IMPLICIT	InitiateUploadSequence-Response,
uploadSegment	[30]	IMPLICIT	UploadSegment-Response,
terminateUploadSequence	[31]	IMPLICIT	TerminateUploadSequence-Response,
requestDomainDownload	[32]	IMPLICIT	RequestDomainDownload-Response,
requestDomainUpload	[33]	IMPLICIT	RequestDomainUpload-Response,
loadDomainContent	[34]	IMPLICIT	LoadDomainContent-Response,
storeDomainContent	[35]	IMPLICIT	StoreDomainContent-Response,
deleteDomain	[36]	IMPLICIT	DeleteDomain-Response,
getDomainAttributes	[37]	IMPLICIT	GetDomainAttributes-Response,
createProgramInvocation	[38]	IMPLICIT	CreateProgramInvocation-Response,
deleteProgramInvocation	[39]	IMPLICIT	DeleteProgramInvocation-Response,
start	[40]	IMPLICIT	Start-Response,
stop	[41]	IMPLICIT	Stop-Response,
resume	[42]	IMPLICIT	Resume-Response,
reset	[43]	IMPLICIT	Reset-Response,
kill	[44]	IMPLICIT	Kill-Response,
getProgramInvocationAttributes	[45]	IMPLICIT	GetProgramInvocationAttributes-Response,
obtainFile	[46]	IMPLICIT	ObtainFile-Response,
defineEventCondition	[47]	IMPLICIT	DefineEventCondition-Response,
deleteEventCondition	[48]	IMPLICIT	DeleteEventCondition-Response,
getEventConditionAttributes	[49]	IMPLICIT	GetEventConditionAttributes-Response,
reportEventConditionStatus	[50]	IMPLICIT	ReportEventConditionStatus-Response,
alterEventConditionMonitoring	[51]	IMPLICIT	AlterEventConditionMonitoring-Response,
triggerEvent	[52]	IMPLICIT	TriggerEvent-Response,
defineEventAction	[53]	IMPLICIT	DefineEventAction-Response,
deleteEventAction	[54]	IMPLICIT	DeleteEventAction-Response,
getEventActionAttributes	[55]	IMPLICIT	GetEventActionAttributes-Response,
reportEventActionStatus	[56]	IMPLICIT	ReportEventActionStatus-Response,
defineEventEnrollment	[57]	IMPLICIT	DefineEventEnrollment-Response,
deleteEventEnrollment	[58]	IMPLICIT	DeleteEventEnrollment-Response,
alterEventEnrollment	[59]	IMPLICIT	AlterEventEnrollment-Response,
reportEventEnrollmentStatus	[60]	IMPLICIT	ReportEventEnrollmentStatus-Response,
getEventEnrollmentAttributes	[61]	IMPLICIT	GetEventEnrollmentAttributes-Response,
acknowledgeEventNotification	[62]	IMPLICIT	AcknowledgeEventNotification-Response,
getAlarmSummary	[63]	IMPLICIT	GetAlarmSummary-Response,
getAlarmEnrollmentSummary	[64]	IMPLICIT	GetAlarmEnrollmentSummary-Response,
readJournal	[65]	IMPLICIT	ReadJournal-Response,
writeJournal	[66]	IMPLICIT	WriteJournal-Response,
initializeJournal	[67]	IMPLICIT	InitializeJournal-Response,
reportJournalStatus	[68]	IMPLICIT	ReportJournalStatus-Response,
createJournal	[69]	IMPLICIT	CreateJournal-Response,

```

deleteJournal          [70] IMPLICIT DeleteJournal-Response,
getCapabilityList       [71] IMPLICIT GetCapabilityList-Response,
fileOpen               [72] IMPLICIT FileOpen-Response,
fileRead               [73] IMPLICIT FileRead-Response,
fileClose              [74] IMPLICIT FileClose-Response,
fileRename             [75] IMPLICIT FileRename-Response,
fileDelete             [76] IMPLICIT FileDelete-Response,
fileDirectory          [77] IMPLICIT FileDirectory-Response,
additionalService      [78] AdditionalService-Response
    -- choices [72] through [77] are reserved for use by services
    -- defined in annex C
    -- choice [79] is reserved
}

```

```

CS-Response-Detail ::= CHOICE {
    status                [0] IMPLICIT CS-Status-Response,
    getNameList           [1] IMPLICIT CS-GetNameList-Response,
    input                 [17] IMPLICIT CS-Input-Response,
    output                [18] IMPLICIT CS-Output-Response,
    initiateDownloadSequence [26] IMPLICIT
        CS-InitiateDownloadSequence-Response,
    downloadSegment       [27] IMPLICIT CS-DownloadSegment-Response,
    terminateDownloadSequence [28] IMPLICIT
        CS-TerminateDownloadSequence-Response,
    initiateUploadSequence [29] IMPLICIT
        CS-InitiateUploadSequence-Response,
    uploadSegment         [30] IMPLICIT CS-UploadSegment-Response,
    terminateUploadSequence [31] IMPLICIT
        CS-TerminateUploadSequence-Response,
    requestDomainDownload [32] IMPLICIT
        CS-RequestDomainDownload-Response,
    requestDomainUpload   [33] IMPLICIT CS-RequestDomainUpload-Response,
    loadDomainContent     [34] IMPLICIT CS-LoadDomainContent-Response,
    storeDomainContent    [35] IMPLICIT CS-StoreDomainContent-Response,
    deleteDomain          [36] IMPLICIT CS-DeleteDomain-Response,
    getDomainAttributes   [37] IMPLICIT CS-GetDomainAttributes-Response,
    createProgramInvocation [38] IMPLICIT
        CS-CreateProgramInvocation-Response,
    deleteProgramInvocation [39] IMPLICIT
        CS-DeleteProgramInvocation-Response,
    start                 [40] IMPLICIT CS-Start-Response,
    stop                  [41] IMPLICIT CS-Stop-Response,
    resume                [42] IMPLICIT CS-Resume-Response,
    reset                 [43] IMPLICIT CS-Reset-Response,
    kill                  [44] IMPLICIT CS-Kill-Response,
    getProgramInvocationAttributes [45] IMPLICIT
        CS-GetProgramInvocationAttributes-Response,
    defineEventCondition  [47] IMPLICIT
        CS-DefineEventCondition-Response,
    deleteEventCondition [48] IMPLICIT
        CS-DeleteEventCondition-Response,
    getEventConditionAttributes [49] IMPLICIT
        CS-GetEventConditionAttributes-Response,
    reportEventConditionStatus [50] IMPLICIT
        CS-ReportEventConditionStatus-Response,
    alterEventConditionMonitoring [51] IMPLICIT
        CS-AlterEventConditionMonitoring-Response,
    triggerEvent          [52] IMPLICIT CS-TriggerEvent-Response,
    defineEventAction     [53] IMPLICIT CS-DefineEventAction-Response,
    deleteEventAction    [54] IMPLICIT CS-DeleteEventAction-Response,
    getEventActionAttributes [55] IMPLICIT

```

reportEventActionStatus	[56]	IMPLICIT	CS-GetEventActionAttributes-Response,
defineEventEnrollment	[57]	IMPLICIT	CS-ReportEventActionStatus-Response,
deleteEventEnrollment	[58]	IMPLICIT	CS-DefineEventEnrollment-Response,
alterEventEnrollment	[59]	IMPLICIT	CS-DeleteEventEnrollment-Response,
reportEventEnrollmentStatus	[60]	IMPLICIT	CS-AlterEventEnrollment-Response,
getEventEnrollmentAttributes	[61]	IMPLICIT	CS-ReportEventEnrollmentStatus-Response,
acknowledgeEventNotification	[62]	IMPLICIT	CS-GetEventEnrollmentAttributes-Response,
getAlarmSummary	[63]	IMPLICIT	CS-AcknowledgeEventNotification-Response,
getAlarmEnrollmentSummary	[64]	IMPLICIT	CS-GetAlarmSummary-Response,
readJournal	[65]	IMPLICIT	CS-GetAlarmEnrollmentSummary-Response,
writeJournal	[66]	IMPLICIT	CS-ReadJournal-Response,
initializeJournal	[67]	IMPLICIT	CS-WriteJournal-Response,
reportJournalStatus	[68]	IMPLICIT	CS-InitializeJournal-Response,
createJournal	[69]	IMPLICIT	CS-ReportJournalStatus-Response,
deleteJournal	[70]	IMPLICIT	CS-CreateJournal-Response,
getCapabilityList	[71]	IMPLICIT	CS-DeleteJournal-Response,
			CS-GetCapabilityList-Response
			}

The ConfirmedServiceResponse parameter shall identify the service type and the response for that service. The context tags provided identify the service type. Definitions for each individual service specify the form of the response for the service through definition of a type, which is referenced by the ConfirmedServiceResponse production.

The AdditionalService-Response shall be reserved for definition of additional responses to services by MMS Companion Standards.

NOTE – All such additional services shall meet the following requirements as described in annex A:

- a) no additional service shall violate any of the state transition rules provided by ISO/IEC 9506-1 and ISO/IEC 9506-2. This limits Companion Standards to definition of substates within existing MMS states;
- b) no additional service shall violate any of the object models provided by ISO/IEC 9506-1 and ISO/IEC 9506-2;
- c) no additional service shall be used to circumvent the spirit and intent of ISO/IEC 9506-1 and ISO/IEC 9506-2.

7.5.5 ServiceError

```

ServiceError ::= SEQUENCE {
  errorClass [0] CHOICE {
    vmd-state [0] IMPLICIT INTEGER { -- VMD-STATE,
      other (0), -- OTHER
      vmd-state-conflict (1), -- VMD-STATE-CONFLICT
      vmd-operational-problem (2), -- VMD-OPERATIONAL-PROBLEM
      domain-transfer-problem (3), -- DOMAIN-TRANSFER-PROBLEM
      state-machine-id-invalid (4) -- STATE-MACHINE-ID-INVALID
    },
    application-reference [1] IMPLICIT INTEGER {
      -- APPLICATION REFERENCE
      other (0), -- OTHER
      application-unreachable (1), -- APPLICATION-UNREACHABLE
      connection-lost (2), -- CONNECTION-LOST
      application-reference-invalid (3), -- APPLICATION-REFERENCE-INVALID
      context-unsupported (4) -- CONTEXT-UNSUPPORTED
    },
    definition [2] IMPLICIT INTEGER { -- DEFINITION
      other (0), -- OTHER
      object-undefined (1), -- OBJECT-UNDEFINED
      invalid-address (2), -- INVALID-ADDRESS
      type-unsupported (3), -- TYPE-UNSUPPORTED
      type-inconsistent (4), -- TYPE-INCONSISTENT
      object-exists (5), -- OBJECT-EXISTS
      object-attribute-inconsistent (6) -- OBJECT-ATTRIBUTE-INCONSISTENT
    },
    resource [3] IMPLICIT INTEGER { -- RESOURCE
      other (0), -- OTHER
      memory-unavailable (1), -- MEMORY-UNAVAILABLE
      processor-resource-unavailable (2), -- PROCESSOR-RESOURCE-UNAVAILABLE
      mass-storage-unavailable (3), -- MASS-STORAGE-UNAVAILABLE
      capability-unavailable (4), -- CAPABILITY-UNAVAILABLE
      capability-unknown (5) -- CAPABILITY-UNKNOWN
    },
    service [4] IMPLICIT INTEGER { -- SERVICE
      other (0), -- OTHER
      primitives-out-of-sequence (1), -- PRIMITIVES-OUT-OF-SEQUENCE
      object-state-conflict (2), -- OBJECT-STATE-CONFLICT
      -- Value 3 reserved for further definition
      continuation-invalid (4), -- CONTINUATION-INVALID
      object-constraint-conflict (5) -- OBJECT-CONSTRAINT-CONFLICT
    },
    service-preempt [5] IMPLICIT INTEGER { -- SERVICE-PREEMPT
      other (0), -- OTHER
      timeout (1), -- TIMEOUT
      deadlock (2), -- DEADLOCK
      cancel (3) -- CANCEL
    },
    time-resolution [6] IMPLICIT INTEGER { -- TIME-RESOLUTION
      other (0), -- OTHER
      unsupportable-time-resolution (1) -- UNSUPPORTABLE-TIME-RESOLUTION
    },
    access [7] IMPLICIT INTEGER { -- ACCESS
      other (0), -- OTHER
      object-access-unsupported (1), -- OBJECT-ACCESS-UNSUPPORTED

```

```

        object-non-existent (2),      -- OBJECT-NON-EXISTENT
        object-access-denied (3),     -- OBJECT-ACCESS-DENIED
        object-invalidated (4)       -- OBJECT-INVALIDATED
    },
    initiate [8] IMPLICIT INTEGER {    -- INITIATE
        other (0),                    -- OTHER
        -- Values 1 and 2 are reserved for further definition
        max-services-outstanding-calling-insufficient (3),
        -- MAX-SERVICES-OUTSTANDING-CALLING-INSUFFICIENT
        max-services-outstanding-called-insufficient (4),
        -- MAX-SERVICES-OUTSTANDING-CALLED-INSUFFICIENT
        service-CBB-insufficient (5),  -- SERVICE-CBB-INSUFFICIENT
        parameter-CBB-insufficient (6),
        -- PARAMETER-CBB-INSUFFICIENT
        nesting-level-insufficient (7)
        -- NESTING-LEVEL-INSUFFICIENT
    },
    conclude [9] IMPLICIT INTEGER {    -- CONCLUDE
        other (0),                    -- OTHER
        further-communication-required (1)
        -- FURTHER-COMMUNICATION-REQUIRED
    },
    cancel [10] IMPLICIT INTEGER {      -- CANCEL
        other (0),                    -- OTHER
        invoke-id-unknown (1),        -- INVOKE-ID-UNKNOWN
        cancel-not-possible (2)       -- CANCEL-NOT-POSSIBLE
    },
    file [11] IMPLICIT INTEGER {        -- FILE
        other (0),                    -- OTHER
        filename-ambiguous (1),       -- FILENAME-AMBIGUOUS
        file-busy (2),                -- FILE-BUSY
        filename-syntax-error (3),    -- FILENAME-SYNTAX-ERROR
        content-type-invalid (4),     -- CONTENT-TYPE-INVALID
        position-invalid (5),         -- POSITION-INVALID
        file-access-denied (6),       -- FILE-ACCESS-DENIED
        file-non-existent (7),        -- FILE-NON-EXISTENT
        duplicate-filename (8),       -- DUPLICATE-FILENAME
        insufficient-space-in-filestore (9)
        -- INSUFFICIENT-SPACE-IN-FILESTORE
    },
    others [12] IMPLICIT INTEGER,      -- OTHERS
    cs-error [13] CS-Service-Error
    -- shall not be chosen in the abstract syntax defined in
    -- Clause 19 of this Part of this International Standard
},
additionalCode [1] IMPLICIT INTEGER OPTIONAL,
additionalDescription [2] IMPLICIT VisibleString OPTIONAL,
serviceSpecificInformation [3] CHOICE {
    obtainFile [0] IMPLICIT ObtainFile-Error,
    start [1] IMPLICIT Start-Error,
    stop [2] IMPLICIT Stop-Error,
    resume [3] IMPLICIT Resume-Error,
    reset [4] IMPLICIT Reset-Error,
    deleteVariableAccess [5] IMPLICIT DeleteVariableAccess-Error,
    deleteNamedVariableList [6] IMPLICIT DeleteNamedVariableList-Error,
    deleteNamedType [7] IMPLICIT DeleteNamedType-Error,
    defineEventEnrollment-Error [8] DefineEventEnrollment-Error,
    fileRename [9] IMPLICIT FileRename-Error,
    -- Reserved for use by annex C Rename service
    additionalService [10] AdditionalService-Error
} OPTIONAL

```

The selection of an `errorClass CHOICE` shall be based on the `Error Class` subparameter of the `ErrorType` parameter specified in the response service primitive. The selection of a value for the `Error Class` selected shall be based on the `Error Code` subparameter of the `ErrorType` parameter specified in the response service primitive. The `additionalCode` and `additionalDescription` parameters shall be derived from the subparameters in the `ErrorType` parameter that have the same name.

NOTE — Service specific information is not specified when a modifier causes the Confirmed-ErrorPDU to be returned. When the Confirmed-ErrorPDU is returned as a result of an error that occurs during the processing of a confirmed service request, the service specific information is returned for those services that provide such information as required by the respective service procedures.

NOTE – All such additional services shall meet the following requirements as described in annex A:

- ## 7.6 Common MMS Types

7.6.1 TimeOfDay

'0000tttttttttttttttttttttttttttttttttttddddd' B

If the value does not contain the Date (four octet content length), the last two octets ("d...d") are omitted. In the bstring representation given above, "t...t" is the relative millisecond of the reported day, with midnight represented by 0, and "d...d" is the relative day, with January 1, 1984 equal to 0. All values are binary values.

The most significant bit of a sub-field value occurs earlier in the bstring with bit significance decreasing for later bits of the bstring.

A system which implements the TimeOfDay type shall specify the granularity of the "t...t" sub-field in the Protocol Implementation Conformance Statement (see clause 18).

7.6.2 Identifiers and Integer Types

The types "Identifier", "Integer8", "Integer16", "Integer32", "Unsigned8", "Unsigned16", and "Unsigned32" are used throughout this part of ISO/IEC 9506. These types are defined as follows.

```
Identifier ::= VisibleString FROM
    ("A"|"a"|"B"|"b"|"C"|"c"|"D"|"d"|"E"|"e"|"F"|"f"|
     "G"|"g"|"H"|"h"|"I"|"i"|"J"|"j"|"K"|"k"|"L"|"l"|
     "M"|"m"|"N"|"n"|"O"|"o"|"P"|"p"|"Q"|"q"|"R"|"r"|
     "S"|"s"|"T"|"t"|"U"|"u"|"V"|"v"|"W"|"w"|"X"|"x"|
     "Y"|"y"|"Z"|"z"|"_"|"0"|"1"|"2"|"3"|"4"|"5"|
     "6"|"7"|"8"|"9") (SIZE(1..32))
    -- An Identifier shall not begin with a digit.

Integer8  ::= INTEGER(-128..127)          -- range -128 <= i <= 127
Integer16 ::= INTEGER(-32768..32767)      -- range -32,768 <= i <= 32,767
Integer32 ::= INTEGER(-2147483648..2147483647)
                                           -- range -2**31 <= i <= 2**31 - 1

Unsigned8  ::= INTEGER(0..127)            -- range 0 <= i <= 127
Unsigned16 ::= INTEGER(0..32767)          -- range 0 <= i <= 32767
Unsigned32 ::= INTEGER(0..2147483647)     -- range 0 <= i <= 2**31 - 1
```

MMS defines various types of names (variable names, type names, etc) in terms of the Identifier production. An Identifier shall be limited in length to 32 octets which shall be chosen from the set characters defined by the VisibleString type as specified in ASN.1 (ISO 8824). Identifiers shall be case sensitive. An Identifier shall not begin with a digit.

Integer8, Integer16, Integer32, Unsigned8, Unsigned16 and Unsigned32 are used throughout this part of ISO/IEC 9506 to represent integers of restricted range, where the minimum and maximum representable values are as specified in the comments following their type declaration.

7.6.3 ObjectName

```
ObjectName ::= CHOICE {
    vmd-specific      [0] IMPLICIT Identifier,
    domain-specific   [1] IMPLICIT SEQUENCE {
        domainID Identifier,
        itemID  Identifier
    },
    aa-specific       [2] IMPLICIT Identifier
}
```

The ObjectName parameter shall be derived according to the rules provided in this part of ISO/IEC 9506, 5.5, using the definition of the ObjectName service parameter defined in ISO/IEC 9506-1, clause 7.

7.6.4 ApplicationReference

```

ApplicationReference ::= SEQUENCE {
    ap-title           [0] ISO-8650-ACSE-1.AP-title OPTIONAL,
    ap-invocation-id   [1] ISO-8650-ACSE-1.AP-invocation-id OPTIONAL,
    ae-qualifier       [2] ISO-8650-ACSE-1.AE-qualifier OPTIONAL,
    ae-invocation-id   [3] ISO-8650-ACSE-1.AE-invocation-id OPTIONAL
}

```

The ApplicationReference parameter shall be derived according to the rules provided in this part of ISO/IEC 9506, 5.5, using the definition of the Application Reference service parameter defined in ISO/IEC 9506-1, clause 6. This ASN.1 definition makes use of the AP-title, AP-invocation-id, AE-qualifier, and AE-invocation-id types defined in the ISO-8650-ACSE-1 module definition provided in ISO 8650.

The values used in the ApplicationReference in any instance of its use shall be chosen such that the ApplicationReference is sufficient to identify uniquely and unambiguously the Application Process, Application Process Invocation, Application Entity, or Application Entity Invocation as required by the referencing MMS service.

NOTE 1 – Additional information on application layer naming and addressing can be found in ISO 7498-3, ISO/IEC 9545, ISO 8649, and ISO 8650.

NOTE 2 – The exact definition and values used for AP-title, AP-invocation-id, AE-qualifier, and AE-invocation-id should be chosen taking into account the ongoing work in the areas of the Association Control Service Element, naming, directories, application layer structure, and the establishment of the Registration Authority for AE-titles.

7.6.5 FileName

```

FileName ::= SEQUENCE OF GraphicString

```

The Filename type shall consist of a sequence of graphic strings. Determination of the semantics of elements in the sequence of graphic strings of a file name shall be a local matter. Any restrictions imposed by a system conforming to ISO/IEC 9506-1 and ISO/IEC 9506-2 on lengths and legal characters in a FileName shall be specified in the Protocol Implementation Conformance Statement (see clause 18). As a minimum, each implementation making use of the filename type defined in this clause shall support filenames containing a single element consisting of one to eight upper case letters or numbers that start with a letter.

NOTE – ISO/IEC 9506-1 and ISO/IEC 9506-2 do not define any interpretation for the components of a filename; such components provide a transparent naming mechanism to the initiator and the responder. The relation between the components defined in the virtual filestore and any division into components in the real system environment is a local implementation choice. An implementation may map a local component structure on to the components of the filename, or it may choose to map its existing filename syntax into a filename with only one component name. An implementation may reflect the MMS filename components in selecting an access path to the real file, but this choice is not in itself visible for interconnection purposes through MMS.

7.6.6 Priority

```

Priority ::= Unsigned8
normalPriority Priority ::= 64

```

The Priority type shall consist of an Unsigned8. The Priority type is an integer with values ranging from zero (0) to one hundred twenty seven (127), inclusive. Zero represents the highest priority. One hundred twenty seven represents the lowest priority. By convention, sixty four (64) represents the "normal" priority.