# Stable Implementation Agreements for Open Systems Interconnection Protocols: Part 20 - Manufacturing Message Specification (MMS)

Output from the December 1993 NIST Workshop for Implementors of OSI

SIG Chair: **Rick Igou, Martin Marietta Energy Systems**
SIG Editor: **Neal Laurance, Ford**

## Foreword

This part of the Stable Implementation Agreements was prepared by the Manufacturing Message Specification (MMS) Special Interest Group (MMSSIG) of the Open Systems Environment Implementors' Workshop (OIW). See Part 1 - Workshop Policies and Procedures of the "Draft Working Implementation Agreements Document" for the charter.

Text in this part has been approved by the Plenary of the above-mentioned Workshop. No significant technical change has occurred in this part since it was previously presented.

Future changes and additions to this version of these Implementor Agreements will be published as change pages. Deleted and replaced text will be shown as strikeout. New and replacement text will be shown as shaded.

# Table of Contents

**Part 20 - Manufacturing Message Specification (MMS)** . . . . . 1

**0 Introduction** . . . . . 1

**1 Scope** . . . . . 1

**2 Field of Application** . . . . . 1
2.1 General . . . . . 1
2.2 Phase 1 agreements . . . . . 1

**3 Normative References** . . . . . 2

**4 Definitions** . . . . . 3

**5 Corrigenda and addenda** . . . . . 3

**6 Status** . . . . . 3

**7 General agreements** . . . . . 3
7.1 Max supported PDU size . . . . . 3
7.2 FileName . . . . . 4

**8 Service-specific agreements** . . . . . 4
8.1 Environment and general management . . . . . 4
8.1.1 Initiate . . . . . 4
8.1.1.1 Negotiation of MMS abstract syntaxes . . . . . 4
8.1.1.2 Max serv outstanding . . . . . 4
8.1.1.3 Local detail calling . . . . . 5
8.1.1.4 Local detail called . . . . . 5
8.1.1.5 Rules of Extensibility . . . . . 5
8.2 VMD support . . . . . 6
8.2.1 Get Capability List service . . . . . 6
8.3 Domain management . . . . . 6
8.3.1 List of capabilities . . . . . 6
8.3.2 Initiate Download Sequence service . . . . . 6
8.3.3 Download Segment service . . . . . 6
8.3.4 Terminate Download Sequence service . . . . . 6
8.3.5 Initiate Upload Sequence service . . . . . 7
8.3.6 Upload Segment service . . . . . 7
8.3.7 Get Domain Attributes service . . . . . 7
8.4 Program Invocation management . . . . . 7
8.4.1 Start service . . . . . 7
8.4.2 Stop service . . . . . 7
8.4.3 Resume service . . . . . 7
8.4.4 Reset service . . . . . 7

8.5 Variable access . . . . . 8
8.5.1 Scattered access . . . . . 8
8.5.2 Floating point . . . . . 8
8.5.3 Unsigned Data . . . . . 8
8.5.4 Order of variable specifications . . . . . 8
8.5.5 Parameter CBBs . . . . . 8
8.5.6 Named Variable Scope . . . . . 9
8.5.7 Address Types . . . . . 9
8.6 Semaphore management . . . . . 9
8.7 Operator communication . . . . . 9
8.8 Event management . . . . . 9
8.9 Journal management . . . . . 9

**Annex A** (normative)

**Backwards compatibility agreements** . . . . . 10
A.1 Introduction . . . . . 10
A.2 Backwards compatibility agreements for calling MMS implementations . . . . . 11
A.3 Backwards compatibility agreements for called MMS implementations . . . . . 11
A.4 General backwards compatibility agreements . . . . . 12
A.4.1 VMD logical status . . . . . 12

**Annex B** (normative)

**DIS 9506 Modifications Required for Backwards Compatibility** . . . . . 13
B.1 Introduction . . . . . 13
B.2 References . . . . . 13
B.3 General . . . . . 13
B.3.1 Implementation base . . . . . 13
B.3.2 Rules of extensibility . . . . . 13
B.4 Modifications to the protocol definitions . . . . . 13
B.4.1 Page 39, Section 7.5.2 of DIS 9506–2 . . . . . 13
B.4.2 Page 49, Section 7.6.4, DIS 9506–2 . . . . . 14
B.4.3 Page 95, Section 12.2.1 of DIS 9506–2 . . . . . 14
B.4.4 Page 96, Section 12.3.1 of DIS 9506–2 . . . . . 14
B.4.5 Page 98, Section 12.4.2 of DIS 9506–2 . . . . . 15
B.4.6 Page 138, Section 15.14 of DIS 9506–2 . . . . . 15
B.4.7 Page 166, Section 17.10 of DIS 9506–2 . . . . . 15
B.5 Behavioral requirements . . . . . 16
B.5.1 Filenames . . . . . 16
B.5.2 Identify service . . . . . 16
B.5.3 Initiate service . . . . . 16
B.5.3.1 Minimum segment size . . . . . 16
B.5.3.2 Maximum segment size . . . . . 16
B.5.4 Abstract syntax name . . . . . 17
B.5.5 Application context name . . . . . 17
B.5.6 Minor version number . . . . . 17
B.6 Parameter CBB subset . . . . . 17
B.7 Service subset . . . . . 18

**Annex C** (normative)

**Basic functional tests** . . . . . 19
C.1 Introduction . . . . . 19
C.1.1 Test steps . . . . . 20
C.1.2 Connections . . . . . 21
C.1.3 Test suites . . . . . 21
C.1.4 General conventions . . . . . 21
C.1.4.1 <LOCAL> . . . . . 21
C.1.4.2 <sPICS:XXX>,<cPICS:XXX> . . . . . 22
C.1.4.3 <sPIXIT:XXX>,<cPIXIT:XXX> . . . . . 22
C.1.4.4 <TMP:XXX> . . . . . 22
C.1.4.5 <ID> . . . . . 22
C.1.5 PICS/PIXIT . . . . . 22
C.1.5.1 General PICS . . . . . 22
C.1.5.2 Server PICS . . . . . 23
C.1.5.3 Client PICS . . . . . 26
C.1.5.4 Addressing information . . . . . 29
C.1.5.5 Server PIXIT . . . . . 30
C.1.5.5.1 Named variable table . . . . . 30
C.1.5.5.2 Addressed variable table . . . . . 31
C.1.5.5.3 Domain table . . . . . 32
C.1.5.5.4 Program Invocation table . . . . . 33
C.1.5.6 PICS/PIXIT Pro–forma . . . . . 34
C.2 Basic functional tests . . . . . 41
C.2.1 Environment and general management test cases . . . . . 41
C.2.1.1 EGM_INIT_01 . . . . . 41
C.2.1.2 EGM_CONC_01 . . . . . 42
C.2.1.3 EGM_ABRT_01 . . . . . 43
C.2.2 VMD support test cases . . . . . 43
C.2.2.1 VMD_STAT_01 . . . . . 44
C.2.2.2 VMD_STAT_02 . . . . . 45
C.2.2.3 VMD_USTA_01 . . . . . 45
C.2.2.4 VMD_GNAM_01 . . . . . 46
C.2.2.5 VMD_IDEN_01 . . . . . 48
C.2.2.6 Rename test cases . . . . . 49
C.2.2.7 VMD_GCAP_01 . . . . . 49
C.2.3 Domain management test cases . . . . . 50
C.2.3.1 DOM_UPLD_01 . . . . . 50
C.2.3.2 DOM_DNLD_01 . . . . . 52
C.2.3.3 DOM_DELE_01 . . . . . 54
C.2.3.4 DOM_GETA_01 . . . . . 54
C.2.4 Program Invocation test cases . . . . . 55
C.2.4.1 PIM_CREA_01 . . . . . 55
C.2.4.2 PIM_DELE_01 . . . . . 56
C.2.4.3 PIM_STRT_01 . . . . . 57
C.2.4.4 PIM_STOP_01 . . . . . 58
C.2.4.5 PIM_RESM_01 . . . . . 59
C.2.4.6 PIM_REST_01 . . . . . 60

C.2.4.7 PIM_GPIA_01 . . . . . 60
C.2.5 Variable access test cases . . . . . 61
C.2.5.1 VAR_READ_01 . . . . . 61
C.2.5.2 VAR_READ_02 . . . . . 62
C.2.5.3 VAR_READ_03 . . . . . 63
C.2.5.4 VAR_WRIT_01 . . . . . 64
C.2.5.5 VAR_WRIT_02 . . . . . 65
C.2.5.6 VAR_WRIT_03 . . . . . 66
C.2.5.7 VAR_RMWV_01 . . . . . 67
C.2.5.8 VAR_RMWV_02 . . . . . 69
C.2.5.9 VAR_RMWV_03 . . . . . 71
C.2.5.10 VAR_IRPT_01 . . . . . 73
C.2.5.11 VAR_IRPT_02 . . . . . 74
C.2.5.12 VAR_IRPT_03 . . . . . 75
C.2.5.13 VAR_GVAA_01 . . . . . 75
C.2.5.14 VAR_GVAA_02 . . . . . 76
C.2.5.15 VAR_GVAA_03 . . . . . 77
C.2.6 Semaphore management test cases . . . . . 78
C.2.7 Operator communication test cases . . . . . 78
C.2.8 Event management test cases . . . . . 78
C.2.9 Journal management test cases . . . . . 78
C.3 Basic functional test script language . . . . . 78
C.4 References . . . . . 78

# List of Tables

Table 1 - Phase 1 Services . . . . . 2
Table 2 - MMS Service Subset . . . . . 18

# Part 20 - Manufacturing Message Specification (MMS)

# 0 Introduction

This section defines Implementors Agreements based on Manufacturing Message Specification (MMS), as defined in ISO/IEC 9506. ISO/IEC 9506 has two parts. Part 1 defines the Virtual Manufacturing Device (VMD), its subordinate abstract objects, and the services on these objects. Part 2 defines the protocol. Future parts may define companion standards.

MMS, as described in ISO/IEC 9506, is based on the following ISO documents: ACSE Service and Protocol (ISO 8649, ISO 8650), Presentation Service and Protocol (ISO 8822, ISO 8823), ASN.1 Abstract Syntax Notation and Basic Encoding Rules (ISO 8824, ISO 8825), and Session Service and Protocol (ISO 8326, ISO 8327). These services and protocols are defined architecturally in the OSI Reference Model (ISO 7498). These agreements provide detailed guidance for the implementor, and eliminate ambiguities in interpretations.

An A-Profile based on MMS and these agreements can be used over any T-Profile (see ISO TR 10000) specifying the OSI connection-mode transport service, or the transport profiles used in support of MAP (Manufacturing Automation Protocol), TOP (Technical and Office Protocols), or US GOSIP.

# 1 Scope

# 2 Field of Application

## 2.1 General

Work on implementation agreements will proceed in phases based upon grouping of services and/or contexts. Implementations are not constrained from implementing services or contexts not addressed by the current set of stable agreements. Future phases of work may affect such implementations.

## 2.2 Phase 1 agreements

These agreements will be based on a subset of MMS services and protocol listed in table 1 and defined in ISO/IEC 9506-1 and ISO/IEC 9506-2.

**Table 1 - Phase 1 Services**

| |
|---|
| Initiate |
| Conclude |
| Reject |
| Abort |
| |
| Status |
| GetNameList |
| Identify |
| UnsolicitedStatus |
| GetCapabilityList |
| |
| InitiateDownloadSequence |
| DownloadSegment |
| TerminateDownloadSequence |
| InitiateUploadSequence |
| UploadSegment |
| TerminateUploadSequence |
| DeleteDomain |
| GetDomainAttributes |
| |
| Read |
| Write |
| InformationReport |
| GetVariableAccessAttributes |
| |
| Input |
| Output |
| |
| CreateProgramInvocation |
| DeleteProgramInvocation |
| Start |
| Stop |
| Resume |
| Reset |
| Kill |
| GetProgramInvocationAttributes |

# 3 Normative References

[**1**] ISO/IEC 9506-1: 1990 - *Industrial automation systems - Manufacturing Message Specification Part 1: Service definition*

[**2**] ISO/IEC 9506-2: 1990 - *Industrial automation systems - Manufacturing Message Specification Part 2: Protocol specification*

[**3**] ISO/IEC 9506-1:1993 - *Industrial automation systems - Manufacturing Message Specification: Technical Corrigenda 1*

# 4 Definitions

The definitions given in ISO/IEC 9506-1 are applicable to this document.

In addition the following definitions are used in this document:

**MMS Implementation** a term used to describe a system which conforms to ISO/IEC 9506, acting either as client or server, when it is unnecessary to distinguish between the MMS-user and the MMS-provider.

**MMSpdu** Any valid value of the MMSpdu abstract data type as defined in Clause 7 of ISO/IEC 9506-2, except for the initiate-RequestPDU, initiate-ResponsePDU, and initiate-ErrorPDU choices, encoded in the negotiated transfer syntax.

# 5 Corrigenda and addenda

(Refer to Working Agreements.)

# 6 Status

(Refer to Working Agreements.)

# 7 General agreements

## 7.1 Max supported PDU size

The max_mms_pdu_size is defined as the maximum number of octets in an MMSpdu encoded using the negotiated transfer syntax. This size shall apply to all MMSpdu's with the exception of the initiate-Request PDU, initiate-Response PDU, and initiate-Error PDU. The max_mms_pdu_size shall be negotiated during connection initiation using the Local Detail Calling and Local Detail Called parameters of the MMS initiate service.

The negotiated max_mms_pdu_size shall be applied as follows:

a) Any received MMSpdu whose length is less than or equal to the negotiated max_mms_pdu_size shall be properly parsed and processed.

b) An MMS implementation should not send an MMSpdu whose size exceeds the negotiated max_mms_pdu_size. If an MMS implementation sends an MMSpdu that exceeds the negotiated max_mms_pdu_size, then it shall be prepared to receive a reject pdu. Should an MMS implementation receive an MMSpdu that exceeds the negotiated max_mms_pdu_size, it shall either reject the MMSpdu or accept the MMSpdu as if no size violation had occurred and perform the expected processing.

c) If an MMS implementation is unable to send a service response because the response would exceed the max_mms_pdu_size, then it shall return a Service response (-) with an error class of SERVICE and an error code of OTHER.

d) When rejecting an MMSpdu because it exceeds the negotiated max_mms_pdu_size, an MMS implementation shall use a Reject PDU Type of PDU-ERROR and a Reject Code of INVALID-PDU in the resulting reject pdu.

## 7.2 FileName

Restrictions for the use of the type FileName in the MMS Abstract Syntax are specified in section 9.1 of part 9 of these agreements.

# 8 Service-specific agreements

## 8.1 Environment and general management

### 8.1.1 Initiate

#### 8.1.1.1 Negotiation of MMS abstract syntaxes

On the A-ASSOCIATE response, the MMS responder shall not accept more than one presentation context derived from an MMS abstract syntax. For this agreement, the term "MMS abstract syntax" shall represent an abstract syntax from the set containing the abstract syntax defined in clause 19 of ISO/IEC 9506-2 and abstract syntaxes defined by MMS companion standards.

**NOTE -** There are technical problems with describing operation in multiple MMS abstract syntaxes over a single association. These problems have been identified as of 9/90, and form the basis of the prior agreement.

#### 8.1.1.2 Max serv outstanding

An MMS Implementation which intends to conform only with the Client Conformance Requirements for Requester CBBs shall:

a) propose one or greater for the value of the Proposed Max Serv Outstanding Called parameter in the Initiate service when initiating the application association (calling);

b) offer one or greater for the value of the Negotiated Max Serv Outstanding Calling parameter in the Initiate service when receiving the application association initiation (called).

An MMS Implementation which intends to conform to one or more Server Conformance Requirements for Responder CBBs shall:

a) propose one or greater for the value of the Proposed Max Serv Outstanding Calling parameter in the Initiate service when initiating the application association (calling);

b) offer one or greater for the value of the Negotiated Max Serv Outstanding Called parameter in the Initiate service when receiving the application association initiation (called).

#### 8.1.1.3 Local detail calling

The local detail calling parameter in the initiate request primitive shall specify the max_mms_pdu_size guaranteed to be supported by the calling MMS implementation. If the local detail calling parameter is absent from the request primitive, then the calling MMS implementation guarantees support for an unlimited max_mms_pdu_size.

If present in the request or indication primitives, the local detail calling parameter shall not be less than 64; however, it is recommended that at least 512 octets be supported.

#### 8.1.1.4 Local detail called

The local detail called parameter in the initiate response shall specify the negotiated max_mms_pdu_size for the application association.

If the local detail calling parameter was omitted in the indication primitive, then the local detail called parameter:

a) may be omitted from the response, indicating that the calling MMS implementation and the called MMS implementation are prepared to support an unbounded max_mms_pdu_size;

b) may be specified in the response, indicating a requirement to support the specified value for max_mms_pdu_size.

If the local detail calling parameter was included in the request, then this parameter shall be present in the response and its value shall be less than or equal to the value of the local detail calling parameter of the request.

If present in the response, the local detail called parameter shall not be less than 64; however, it is recommended that at least 512 octets be supported.

#### 8.1.1.5 Rules of Extensibility

Any additional valid tagged ASN.1 values received as sequence elements in the parameters of the Initiate-RequestPDU, the Initiate-ResponsePDU, or the Initiate-ErrorPDU shall be ignored for upward compatibility purposes.

Implementations shall be capable of parsing up to 128 bits in the services supported field of either the Initiate-RequestPDU or the Initiate-ResponsePDU. Implementations shall be capable of parsing up to 32 bits in the parameter CBB field of either the Initiate-Request-PDU or the Initiate-ResponsePDU. In both

cases, the behaviour of the implementation shall be no different than if the PDU received had not contained additional bits.

## 8.2 VMD support

### 8.2.1 Get Capability List service

Only one capability shall be described in each VisibleString of the SEQUENCE OF.

## 8.3 Domain management

### 8.3.1 List of capabilities

Only one capability shall be described in each Visible String of the SEQUENCE OF.

The order of the strings within the List of Capabilities may have significance to the server implementation, and the order shall be preserved.

### 8.3.2 Initiate Download Sequence service

The List of Capability parameter shall follow the limitations of 8.3.1.

The syntax and semantics of the capabilities shall be defined by the Server in the PICS. Any deviation from the defined syntax and semantics shall be grounds for the Server to return a service error with Error Class = RESOURCE and Error Code = CAPABILITY-UNKNOWN.

### 8.3.3 Download Segment service

A client that receives a Download Segment indication after issuing a Download Segment Result(+) with the MoreFollows parameter equal to FALSE or after issuing a Download Segment Result(-) shall issue either a service error, specifying an Error Class = SERVICE and an Error Code = PRIMITIVES-OUT-OF-SEQUENCE, or an Abort request.

### 8.3.4 Terminate Download Sequence service

If a client receives a Terminate Download Sequence indication in which the Discard parameter is absent and the client has not issued a Download Segment response with the More Follows parameter = FALSE for that Domain, it shall behave as if it had received a Terminate Download Sequence indication with the Discard parameter present with error class = VMD-STATE and error code = DOMAIN-TRANSFER-PROBLEM. It is then up to the client application to determine the true state of the Domain and take any recovery action.

### 8.3.5 Initiate Upload Sequence service

The List of Capability parameter shall follow the limitations of 8.3.1.

### 8.3.6 Upload Segment service

A server that receives an Upload Segment indication for an Upload State Machine for which it has issued an Upload Segment Result(-) or an Upload Segment Result(+) with the MoreFollows parameter equal to FALSE, shall issue either a service error, specifying an Error Class = SERVICE and an Error Code = PRIMITIVES-OUT-OF-SEQUENCE, or an Abort request.

### 8.3.7 Get Domain Attributes service

The List of Capability parameter shall follow the limitations of 8.3.1.

## 8.4 Program Invocation management

### 8.4.1 Start service

A ProgramInvocationState of non-existent shall be returned in a Result(-) when a request to Start a non-existent Program Invocation is received.

### 8.4.2 Stop service

A ProgramInvocationState of non-existent shall be returned in a Result(-) when a request to Stop a non-existent Program Invocation is received.

### 8.4.3 Resume service

A ProgramInvocationState of non-existent shall be returned in a Result(-) when a request to Resume a non-existent Program Invocation is received.

### 8.4.4 Reset service

A ProgramInvocationState of non-existent shall be returned in a Result(-) when a request to Reset a non-existent Program Invocation is received.

## 8.5 Variable access

### 8.5.1 Scattered access

It is strongly recommended that for services which use variable access, a Variable List Name or List of Variable be used instead of Scattered Access.

No implementations shall be required to propose or accept the VSCA Parameter CBB.

### 8.5.2 Floating point

It is strongly recommended for services which use floating point types or values, that the choice of floating-point in the Data and TypeSpecification productions be used instead of the real choice.

No implementations shall be required to propose or accept the REAL parameter CBB.

Any implementation which supports data of the MMS Floating Point Type, shall be capable of supporting a size parameter of format width 32 and exponent width 8.

### 8.5.3 Unsigned Data

Upon receipt of an MMSpdu containing a negative value for either an unsigned choice or a bcd choice in the Data production, an implementation shall treat this occurrence as a protocol error.

### 8.5.4 Order of variable specifications

The order of variable specifications that appear in lists shall not constrain the temporal order of the access to indivicual variables by the V-Put and V-Get functions in the server.

### 8.5.5 Parameter CBBs

Each server implementation that claims support for the Read, Write or InformationReport service shall be capable of supporting either the VNAM or VADR parameter CBB.

Each client implementation that claims support for the Read, Write or InformationReport service shall be capable of supporting the VNAM and VADR parameter CBBs.

### 8.5.6 Named Variable Scope

Each server implementation that claims support for the VNAM parameter CBB shall be capable of supporting either VMD-Specific or Domain-Specific named variables.

Each client implementation that claims support for the VNAM parameter CBB shall be capable of supporting both VMD-Specific and Domain-Specific named variables.

### 8.5.7 Address Types

Each server implementation that claims support for the VADR parameter CBB shall be capable of supporting either the Symbolic-Address or Numeric-Address choice.

Each client implementation that claims support for the VADR parameter CBB shall be capable of supporting both the Symbolic-Address and Numeric-Address choices.

## 8.6 Semaphore management

Semaphore services are not considered in Phase 1.

## 8.7 Operator communication

No Operator Communication agreements have been identified to date.

## 8.8 Event management

Event Management services are not considered in Phase 1.

## 8.9 Journal management

Journal Management services are not considered in Phase 1.

# Annex A (normative)

# Backwards compatibility agreements

## A.1 Introduction

There is an installed base of real DIS 9506 based implementations. Providing support for application connectivity to both DIS and IS is desired as a migration strategy. These implementation agreements will allow IS based implementations to interoperate with DIS based implementations as described in ANNEX B. To achieve this backwards compatibility, the IS implementation shall support all of the agreements in this section.

It was found that the Abstract Syntax name object identifiers of both DIS and IS were identical. Therefore, the use of zero as the version number allows differentiation between an IS and a DIS based implementation. Since the abstract syntax name object identifier of any companion standard is different from that used by the DIS implementations, DIS implementations cannot interoperate with IS based implementations in a companion standard context.

**NOTES**

1 The value of zero is a valid value for this parameter in the DIS and not in the IS.

2 There are three types of implementations when considering MMS backwards compatibility.

**IMP-1:** An implementation based on DIS 9506 as described in Annex B;

**IMP-2:** An implementation based on IS 9506 with no backwards compatibility agreements applied;

**IMP-3:** An implementation based on IS 9506 which includes the backwards compatibility agreements of this annex. Such an implementation can dynamically negotiate to interoperate with an IMP-1, an IMP-2, or an IMP-3 implementation.

Since the value of the minor version number is zero for DIS-based implementations, and is one or greater for implementations of ISO/IEC 9506, this value can be used to differentiate between IMP-1 and IMP-2. An IMP-3 system can interoperate with either of these systems. If an IMP-3 is the Calling system, it will offer a value of one (or greater) for the proposed version number. An IMP-1 system will respond with a value of the negotiated version number of zero, using the negotiation procedure defined in ISO/IEC 9506. The IMP-3 system will accept this response. If the IMP-3 system is the Called system and has received an Initiate request with a value of zero for the proposed version number (from an IMP-1 system), it will respond with a value of zero for the negotiated version number. By following this procedure, an IMP-3 can interoperate with an IMP-2 or with another IMP-3 viewed as an IMP-2. After association context establishment, an IMP-3 system shall behave as either an IMP-1 or an IMP-2 system as appropriate on that particular association. The remainder of this section describes additional agreements which change an IMP-2 implementation into an IMP-3 implementation.

## A.2 Backwards compatibility agreements for calling MMS implementations

A calling MMS implementation shall be capable of receiving and supporting a negotiatedVersionNumber parameter in the Initiate Service confirm of zero.

A calling MMS implementation which has received a negotiatedVersionNumber parameter in the Initiate Service confirm of zero shall support the modifications described in A.4.

A calling MMS implementation shall be capable of receiving an Application Context Name parameter value appropriate to an IMP-1 or IMP-2 in the A-Associate confirm.

A calling MMS implementation which has received a negotiatedVersionNumber of zero shall be capable of receiving and supporting an Initiate Response which has been encoded according to the modifications described in Appendix B, specifically the capability of receiving and supporting a negotiatedParameterCBB containing exactly 7 bits.

If a calling MMS implementation receives an Initiate confirm primitive with a negotiatedVersionNumber parameter equal to zero, the calling MMS implementation shall support the VLIS conformance building block if the implementation claims support for any service which contains one or more parameters which indicate the VLIS CBB in its service definition.

## A.3 Backwards compatibility agreements for called MMS implementations

A called MMS implementation shall be capable of receiving and supporting a proposedVersionNumber parameter in the Initiate Service indication of zero.

A called MMS implementation which has received a proposedVersionNumber parameter in the Initiate Service indication of zero shall support the modifications in A.4.

A called MMS implementation shall be capable of receiving an Application Context Name parameter appropriate to an IMP-1 or IMP-2 in the A-Associate indication.

A called MMS implementation shall be capable of receiving and supporting an Initiate Request which has been encoded according to the modifications described in Appendix B, specifically the capability of receiving and supporting a proposedParameterCBB containing exactly 7 bits.

If a called MMS implementation receives an Initiate indication primitive with a proposedVersionNumber parameter equal to zero, the called MMS implementation shall support the VLIS conformance building block if the implementation claims support for any service which contains one or more parameters which indicate the VLIS CBB in its service definition.

## A.4 General backwards compatibility agreements

### A.4.1 VMD logical status

If the current VMD State is SUPPORT-SERVICES-ALLOWED and the association minor version number is zero, then the vmdLogicalStatus parameter shall have a value of STATE-CHANGES-ALLOWED in a Status response or in an unsolicitedStatus request.

# Annex B (normative)

# DIS 9506 Modifications Required for Backwards Compatibility

## B.1 Introduction

This annex is an integral part of this part. It documents the modifications to DIS 9506 required to describe implementations for which the agreements of this part provide backwards compatibility. This annex as applied to DIS 9506 is referred to as Version 0.

## B.2 References

[**1**] *MMS/1 Manufacturing Message Specification - ISO DIS 9506 - Service Definition*, December 1987

[**2**] *MMS/2 Manufacturing Message Specification - ISO DIS 9506 - Protocol Specification*, December 1987

[**3**] *NBS OSI Implementors Workshop Agreements* - December 1987

## B.3 General

### B.3.1 Implementation base

Version 0 is based upon Reference [3] in B.2 as it applies to MMS.

### B.3.2 Rules of extensibility

The following sentence is appended to the last paragraph in section 8.2.1.1.5.2 Proposed Parameter CBB and the last paragraph in section 8.2.1.2.5.2 Negotiated Parameter CBB of DIS 9506-1.

"Any additional bits shall be ignored."

## B.4 Modifications to the protocol definitions

### B.4.1 Page 39, Section 7.5.2 of DIS 9506-2

```
CHANGE

 reportEventEnrollmentStatus   [60] IMPLICIT
                                    ReportEventEnrollmentStatus-Request,

TO

 reportEventEnrollmentStatus   [60] ReportEventEnrollmentStatus-Request,
```

### B.4.2 Page 49, Section 7.6.4, DIS 9506-2

```
CHANGE

 ApplicationReference  ::= SEQUENCE {
   ap-title                ISO-8650-ACSE-1.AP-title OPTIONAL,
   ap-invocation-id        ISO-86 50-ACSE-1.AP-invocation-id OPTIONAL,
   ae-qualifier            ISO-8650-ACSE-1.AE-qualifier OPTIONAL,
   ae-invocation-id        ISO-8650-ACSE-1.AE-invocation-id OPTIONAL }

TO

 ApplicationReference  ::= SEQUENCE {
   ap-title                [0] OBJECT IDENTIFIER OPTIONAL,
   ap-invocation-id        [1] INTEGER OPTIONAL,
   ae-qualifier            [2] INTEGER OPTIONAL,
   ae-invocation-id        [3] INTEGER OPTIONAL }
```

### B.4.3 Page 95, Section 12.2.1 of DIS 9506-2

```
CHANGE

 structure   [2] IMPLICIT SEQUENCE OF SEQUENCE {

TO

 structure   [2] IMPLICIT SEQUENCE {
```

### B.4.4 Page 96, Section 12.3.1 of DIS 9506-2

| CHANGE |
| --- |
| named [4] IMPLICIT SEQUENCE { |
| TO |
| named [5] IMPLICIT SEQUENCE { |

### B.4.5 Page 98, Section 12.4.2 of DIS 9506-2

| CHANGE |
| --- |
| generalized-time [10] IMPLICIT GeneralizedTime, |
| TO |
| generalized-time [11] IMPLICIT GeneralizedTime, |

### B.4.6 Page 138, Section 15.14 of DIS 9506-2

| CHANGE |
| --- |
| additionalDetail [9] IMPLICIT EE-Additional-Detail OPTIONAL |
| TO |
| additionalDetail [9] EE-Additional-Detail OPTIONAL |

### B.4.7 Page 166, Section 17.10 of DIS 9506-2

| CHANGE the transfer syntax object identifier value from |
| --- |
| { iso asn1(1) basic-encoding(1) } |
| TO |
| { joint-iso-ccitt asn1(1) basic-encoding(1) } |

## B.5 Behavioral requirements

### B.5.1 Filenames

File Names are specified in accordance with the NBS Implementors' agreements for FTAM Reference [3] in B.2.

### B.5.2 Identify service

In the Identify service, the vendor, model and revision fields may be of any length, but only the first 64, 16, and 16 octets respectively are treated as significant.

### B.5.3 Initiate service

An MMS Client will:

a) propose 1 or greater for the value of the Proposed Max Serv Outstanding Called parameter in the Initiate service when initiating the application association (calling);

b) offer 1 or greater for the value of the Negotiated Max Serv Outstanding Calling parameter in the Initiate service when receiving the application association initiation (called).

An MMS Server will:

a) propose 1 or greater for the value of the Proposed Max Serv Outstanding Calling parameter in the Initiate service when initiating the application association (calling);

b) offer 1 or greater for the value of the Negotiated Max Serv Outstanding Called parameter in the Initiate service when receiving the application association initiation (called).

#### B.5.3.1 Minimum segment size

MMS implementations are able to parse and process 512 octets of MMSpdu as they are encoded in ASN.1 basic encoding rules.

#### B.5.3.2 Maximum segment size

The Max Segment Size is defined as the maximum number of octets in an MMSpdu encoded using the negotiated transfer syntax. This size will apply to all MMSpdu's with the exception of the initiate-Request PDU, initiate-Response PDU, and the initiate-Error PDU. The max segment size will be negotiated during connection initiation using the Proposed Max Segment Size and Negotiated Max Segment Size parameters of the MMS initiate service.

The Max Segment Size will be applied as follows:

a) Any received MMSpdu which is less than or equal to the Max Segment Size will be properly parsed and processed;

b) An MMS implementation will not send an MMSpdu whose size exceeds the Max Segment Size.

### B.5.4 Abstract syntax name

The ASN.1 object identifier value for the abstract syntax name will be the same as specified on page 166, section 17.10 of DIS 9506-2.

### B.5.5 Application context name

The ASN.1 object identifier value for the application context name will be the same as specified on page 166, section 17.11 of DIS 9506-2.

An MMS implementation ignores the Application Context Name in the A-Associate indication and the A-Associate confirm.

### B.5.6 Minor version number

The Minor Version Number is zero.

## B.6 Parameter CBB subset

The following subset of MMS Parameter CBBs were considered during preparation of this annex:

a) STR1;

b) NEST;

c) VADR;

d) VNAM.

## B.7 Service subset

The following subset of MMS services were considered during preparation of this annex.

**Table 2 - MMS Service Subset**

Initiate
Conclude
Cancel
Status
GetNameList
Identify
UnsolicitedStatus
GetCapabilityList
InitiateDownloadSequence
DownloadSegment
TerminateDownloadSequence
InitiateUploadSequence
UploadSegment
TerminateUploadSequence
RequestDomainDownload
RequestDomainUpload
LoadDomainContent
StoreDomainContent
DeleteDomain
GetDomainAttributes
Read
Write
InformationReport
GetVariableAccessAttributes
Input
Output
TakeControl
RelinquishControl
ReportSemaphoreStatus
ReportPoolSemaphoreStatus
ReportSemaphoreEntryStatus
CreateProgramInvocation
DeleteProgramInvocation
Start
Stop
Resume
Reset
Kill
GetProgramInvocationAttributes
ObtainFile
GetEventConditionAttributes
ReportEventConditionStatus
GetAlarmSummary
ReadJournal
WriteJournal
InitializeJournal
CreateJournal
DeleteJournal
ReportJournalStatus

# Annex C (normative)

# Basic functional tests

## C.1 Introduction

This document defines a set of basic functional test steps which can be used as building blocks in developing various abstract test specification for basic functional tests, interoperability tests, conformance tests, and performance tests. This document also defines a set of basic functional tests based on these tests steps.

MMS provides a total of 86 messaging services which are oriented toward the support of factory floor devices. Examples of MMS services include reading a named variable from a remote data acquisition system, requesting status from a remote device and loading programs into programmable devices. Many of the services interact with one another in complicated ways. The complexity of the protocols makes exhaustive testing impractical on both technical and economic grounds. Further, there is no guarantee that a system which has passed a set of basic functional tests will interoperate with other systems. Rather, passing the tests provides a level of confidence that the system will likely interoperate with other systems and should behave in a consistent manner in representative instances of communications.

The basic functional tests described in this document, do not constitute a complete specification for a test implementation. The tests are intended to be used like building blocks in developing test implementations for a variety of specific purposes. Performance testing, stress testing and acceptance testing are a few instances of the testing variations which may be supported.

The basic functional tests are intended to be used as building blocks for the development of tests which verify the behavior of devices in a manner which can be directly related to their use on factory floors. Several requirements have been placed on the basic functional tests to promote this capability.

The following are the requirements used in the development of the BFT's:

a) The basic functional tests are designed to be highly modular;

b) The basic functional tests are designed to permit both "tester-to-vendor" and "vendor-to-vendor" architectures;

c) The basic functional tests are designed to be implemented without requiring any modification to a vendors product;

d) The basic functional tests are designed to allow "geographical independence". In other words, remote testing is allowed;

e) The basic functional tests strive to be stated in "pass/fail" terms to make test documentation easier;

f) The basic functional tests are designed to test both clients and servers;

g) The basic functional tests are designed to be automated, although this is not required.

### C.1.1 Test steps

Each test step is defined by a combination of six attributes - its name, purpose, inputs, initial conditions, sequence and pass condition.

The naming convention used to uniquely generate a test **name** is as follows. The test name is made up of three components. The first component identifies the service group. The second component identifies the service. The third component identifies the test step number.

Values for the first component are:

EGM - Environment and General Management Services;
VMD - Virtual Manufacturing Device Support Services;
DOM - Domain Management Services;
PIM - Program Invocation Management Services;
VAR - Variable Access Services;
SEM - Semaphore Management Services;
OPR - Operator Communications Services;
EVM - Event Management Services;
JOU - Journal Management Services;
OBF - File Access Service;
FIL - File Management Services;

The second component is always a sequence of four upper case letters. An example would be the name for the status service under the VMD group - STAT. The values for the second component for each service group are listed under the sections for that group.

The third component is used to distinguish between test steps where the first two name components are the same. It is always a two digit decimal number.

As an example, the complete name for the first test step for the status service would be VMD_STAT_01.

The **purpose** briefly describes the capability being tested.

The **inputs** specify necessary information which must be provided in order to run the test. It is not specified how this information is provided because that depends on the service being tested as well as the implementation of a given instance of the test. Typical sources of test input information would be the vendor Protocol Implementation Conformance Statement (PICS), the vendor Protocol Implementation Extra Information Statement (PIXIT) and information supplied by a test operator.

The **initial conditions** state conditions under which the test may be run. Typical conditions would include the requirement that a connection be established, that parameter and service support appropriate for the test be successfully negotiated and any other conditions that are unique to the service being tested.

The **sequence** defines the sequence and form of messages that will appear "on the wire" when the test is executed. The purpose of defining the test sequence from the point of view of a third party observing the

message transaction is that the same test case can support testing an MMS client or server or both.

The **pass condition** states the criteria for determining whether or not a test has been successfully passed. The pass condition that the messages exchanged "on the wire" parse as described under the sequence section is always in effect and is not explicitly stated. It is intended that the pass condition be pass/fail based on parsing of the messages exchanged for a test which has been executed. A test which is not run because it is not applicable is reported as the pass condition "not applicable." A test which is not run because the initial conditions cannot be satisfied is reported as the pass condition "could not run".

## C.1.2 Connections

The ability to establish a connection is basic to the operation of all of the MMS basic functional tests. (The test for initiate itself is the exception.) If the connection requirements as stated in the test initial conditions cannot be provided, the test fails.

In most cases, a single connection can be used to run one or more test cases if the required connection parameters for each test are supported. There is no requirement that a new connection be made in order to run each test case. Some services create state machines and other objects associated with the VMD. In these cases, there may be a dependency on connection state between tests. These dependencies are described in the test documentation.

## C.1.3 Test suites

Test suites based on groupings of the basic functional tests may be developed to satisfy the testing of various device types. The actual test suite shall be based on the services supported, the parameter CBB's supported and local implementation values described in the device PICS.

The tests are designed such that pass conditions can be determined from the point of view of the requesting or responding MMS-user. This allows the same basic functional tests to be used to form a test suite for either a client or a server.

## C.1.4 General conventions

This document describes the basic functional test steps with a notation which is similar to ASN.1. Several extensions are employed which are introduced and terminated by "<" and ">" respectively.

### C.1.4.1 <LOCAL>

When used in the sequence sections of this document, the symbol <LOCAL> indicates a parameter value which is determined by a local action of the sending system.

#### C.1.4.2 <sPICS:XXX>,<cPICS:XXX>

When used in the sequence and pass conditions sections of this document, the symbols <sPICS:XXX> and <cPICS:XXX> indicate parameter values which are determined by a vendor PICS document for the server and client respectively. The PICS, for the purpose of basic functional testing, is assumed to be in a structured machine readable form, and values in the PICS are accessed in a manner similar to structures in the language "C". An example would be the symbol <PICS:VendorName> which references the vendor name field in the PICS document.

#### C.1.4.3 <sPIXIT:XXX>,<cPIXIT:XXX>

When used in the sequence and pass condition sections of this document, the symbols <sPIXIT:XXX> and <cPIXIT:XXX> indicate parameter values which are determined by vendor PIXIT documents for the server and client respectively. The PIXIT, for the purpose of basic functional testing, is assumed to be in a structured machine readable form, and values in the PIXIT are accessed in a manner similar to structures in the language "C".

#### C.1.4.4 <TMP:XXX>

Values of certain PDU parameters will often need to be saved or accumulated in order to evaluate the pass criteria for the test. Sometimes they will also be needed to parse the messages. The text in the sequence sections of this document will clearly state which parameters have to be used for this purpose. An example of a temporary variable is the visibleString <TMP:vendorName> used in VMD_IDENT_01 to save the contents of the vendorName field in the identify service response for evaluating the pass criteria for the identify test.

#### C.1.4.5 <ID>

In general, the exact values of InvokeId's are of no interest in performing or analyzing the tests. Any problems associated with the use of InvokeId's are assumed to be caught by conformance testing. In the sequence section of this document, InvokeId's are simply referred to by the symbol <ID>. They are used in parsing to match a request with a response.

### C.1.5 PICS/PIXIT

This section is intended to represent a machine-readable PICS/PIXIT pro-forma. The format and notations used for each section will be explained in each section.

#### C.1.5.1 General PICS

These are general information that apply to both Client and Server. The appropriate information for the IUT should be inserted where the blanks are shown.

**VendorName** _____

**ModelName** _____

**Revision** _____

### C.1.5.2 Server PICS

The following section defines text which should be listed if a given service is supported by the IUT as a Server, and omitted if the service is not supported. Support of a service is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "s" and concatenating the service name.

sInitiate

sConclude

sAbort

sStatus

sUnsolicitedStatus

sGetNameList

sIdentify

sGetCapabilityList

sDomainUpload

sDomainDownload

sDomainDelete

sGetDomainAttributes

sCreateProgramInvocation

sDeleteProgramInvocation

sStart

sStop

sResume

sReset

sGetProgramInvocationAttributes

sRead

sWrite

sInformationReport

sGetVariableAccessAttributes

The next section of the Server PICS consists of text which represents parameter CBB support. A CBB is listed if it is supported by the IUT as a Server, and omitted if the CBB is not supported. Support of a CBB is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "s" and concatenating the CBB name.

sVADR

sVNAM

sVLIS

sSTR1

sSTR2

The next PICS information which needs to be specified is the acceptable range of the parameter CBB NEST. The two values that need to be specified are:

**sNESTmin** the minimum nesting level allowed to be proposed by the Client such that an MMS context is allowed to be established.

**sNESTmax** the maximum nesting level that can ever be returned in the initiate-ResponsePDU.

An example of specification is: sNESTmin=1.

Similarly, maximum services outstanding calling and called need to be specified in the PICS. The values to be defined are:

**sMaximumServicesOutstandingCallingMin** the minimum value that can be proposed by the Client such that an MMS context is allowed to be established.

**sMaximumServicesOutstandingCallingMax** the maximum value that can ever be returned in the initiate-ResponsePDU.

**sMaximumServicesOutstandingCalledMin** the minimum value that can be proposed by the Client such that an MMS context is allowed to be established.

**sMaximumServicesOutstandingCalledMax** the maximum value that can ever be returned in the initiate-ResponsePDU.

Next the range of local detail calling and called must be specified. The OIW stable agreements specify that

these values are used to negotiate maximum size of an MMSpdu. The values which must be specified are:

**sInitiateLocalDetailMin** the minimum value that can ever be returned in the initiate-ResponsePDU.

**sInitiateLocalDetailMax** the maximum value that can ever be returned in the initiate-ResponsePDU.

Next to be defined is the list of abstract syntaxes supported by the Server. Definition will be via ASN.1 value notation format.

```
sListOfAbstractSyntaxes ::= SEQUENCE OF OBJECT IDENTIFIER
```

Load data format will be specified via ASN.1 value notation.

```
sLoadDataFormat ::= SEQUENCE {
      octet-string NULL OPTIONAL,
      external SEQUENCE OF OBJECT IDENTIFIER OPTIONAL
      }
```

The next section of the Server PICS consists of text which represents the type of address format supported. This section is only to be supplied if sVADR is specified in the parameter CBB portion of the Server PICS. An address format is listed if it is supported by the IUT as a Server, and omitted if it is not supported. Support is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "s" and concatenating the address type name.

sSymbolicAddress

sNumericAddress

sUnconstrainedAddress

The next section of the Server PICS consists of text which represents the primitive data types supported. A data type is listed if it is supported by the IUT as a Server, and omitted if it is not supported. Support is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "s" and concatenating the address type name.

sboolean

sbit-string

sinteger

sunsigned

sfloating-point

sreal

soctet-string

svisible-string

sgeneralized-time

sbinary-time

sbcd

sbooleanArray

sobjId

### C.1.5.3 Client PICS

The following section defines text which should be listed if a given service is supported by the IUT as a Client, and omitted if the service is not supported. Support of a service is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "c" and concatenating the service name.

cInitiate

cConclude

cAbort

cStatus

cUnsolicitedStatus

cGetNameList

cIdentify

cGetCapabilityList

cDomainUpload

cDomainDownload

cDomainDelete

cGetDomainAttributes

cCreateProgramInvocation

cDeleteProgramInvocation

cStart

cStop

cResume

cReset

cGetProgramInvocationAttributes

cRead

cWrite

cInformationReport

cGetVariableAccessAttributes

The next section of the Client PICS consists of text which represents parameter CBB support. A CBB is listed if it is supported by the IUT as a Client, and omitted if the CBB is not supported. Support of a CBB is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "c" and concatenating the CBB name.

cVADR

cVNAM

cVLIS

cSTR1

cSTR2

The next PICS information which needs to be specified is the acceptable range of the parameter CBB NEST. The two values that need to be specified are:

**cNESTmax** the maximum nesting level which will be proposed by the Client in the initiate-RequestPDU.

**cNESTmin** the minimum nesting level that can be returned in the initiate-ResponsePDU without causing the loss of the MMS context.

An example of specification is: cNESTmin=1.

Similarly, maximum services outstanding calling and called need to be specified in the PICS. The values to be defined are:

**cMaximumServicesOutstandingCallingMax** the maximum value that will be proposed by the Client in the initiate-RequestPDU.

**cMaximumServicesOutstandingCallingMin** the minimum value that can be returned in the initiate-ResponsePDU without causing the loss of the MMS context.

**cMaximumServicesOutstandingCalledMax** the maximum value that will be proposed by the

Client in the initiate-RequestPDU.

**cMaximumServicesOutstandingCalledMin** the minimum value that can be returned in the initiate-ResponsePDU without causing the loss of the MMS context.

Next the range of local detail calling and called must be specified. The OIW stable agreements specify that these values are used to negotiate maximum size of an MMSpdu. The values which must be specified are:

**cInitiateLocalDetailMax** the maximum value that will be proposed in the initiate-RequestPDU.

**cInitiateLocalDetailMin** the minimum value that can ever be returned in the initiate-ResponsePDU without causing the loss of the MMS context.

Next to be defined is the list of abstract syntaxes supported by the Client. Definition will be via ASN.1 value notation format.

```
cListOfAbstractSyntaxes ::= SEQUENCE OF OBJECT IDENTIFIER
```

Load data format will be specified via ASN.1 value notation.

```
cLoadDataFormat ::= SEQUENCE {
      octet-string NULL OPTIONAL,
      external SEQUENCE OF OBJECT IDENTIFIER OPTIONAL
      }
```

The next section of the Client PICS consists of text which represents the type of address format supported. This section is only to be supplied if cVADR is specified in the parameter CBB portion of the Client PICS. An address format is listed if it is supported by the IUT as a Client, and omitted if it is not supported. Support is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "c" and concatenating the address type name.

cSymbolicAddress

cNumericAddress

cUnconstrainedAddress

The next section of the Client PICS consists of text which represents the primitive data types supported. A data type is listed if it is supported by the IUT as a Client, and omitted if it is not supported. Support is defined in ISO/IEC 9506. The list of acceptable values are constructed by taking the letter "c" and concatenating the address type name.

cboolean

cbit-string

cinteger

cunsigned

cfloating-point

creal

coctet-string

cvisible-string

cgeneralized-time

cbinary-time

cbcd

cbooleanArray

cobjId

### C.1.5.4 Addressing information

This is general information that is needed by both Client and Server.

ServerDeviceConnectionData

sAE-Title=

sPSAP=

sSSAP=

sTSAP=

sNSAP=

sMAC=

sLSAP=

ClientDeviceConnectionData

cAE-Title=

cPSAP=

cSSAP=

cTSAP=

cNSAP=

cMAC=

cLSAP=

### C.1.5.5 Server PIXIT

#### C.1.5.5.1 Named variable table

The list of supported named variables is defined via the following ASN.1 definitions. ASN.1 value notation shall be used for the actual definitions.

```
namedVariableTable ::= SEQUENCE OF NamedVariableTableEntry

      NamedVariableTableEntry ::= SEQUENCE  {
        index        [0] INTEGER,
        name         [1] ObjectName,
        typeSpec     [2] TypeSpecification,
        data          [3] Data OPTIONAL,-- Field is only present if a specific
                        -- data value is required for variable write tests.
                   -- If not present, it is assumed that any data value
                        -- consistent with the type specification is adequate.
        access       [4] VisibleString      -- R, W, V
        address      [5] Address OPTIONAL,      -- present if PUBLIC
                                                -- not present otherwise
        predefined   [6] BOOLEAN,
        mmsDeletable [7] BOOLEAN
      }
```

Examples of the definitions follow.

```
namedVariableTableEntry {
index 0,
name {vmd-specific "PART_COUNT"},
typeSpec {integer '32'},
data {integer '14'},
access "RWV",
address {symbolicAddress "$N0:00"},
predefined FALSE,
mmsDeletable TRUE
}
```

```
namedVariableTableEntry {
index 1,
name {Domain-specific {DomainID 'DOM1', itemID 'TEMP'}},
typeSpec {unsigned '16'},
data {integer '1600'},
access "R",
address {symbolicAddress "R125"},
predefined TRUE,
mmsDeletable FALSE
}

namedVariableTableEntry {
index 2,
name {vmd-specific "ARRAY"},
typeSpec {array {packed TRUE,
                 numberOfElements '100',
                 elementType {integer '16'}
                }},
      -- data not present
access "W",
address {unconstrainedAddress 0x12345678},
predefined TRUE,
mmsDeletable FALSE
}
```

#### C.1.5.5.2 Addressed variable table

The list of supported addressed variables is defined via the following ASN.1 definitions. ASN.1 value notation shall be used for the actual definitions.

```
  AddressVariableTable ::= SEQUENCE OF AddressVariableTableEntry

      AddressVariableTableEntry ::= SEQUENCE {
        index          [0] INTEGER,
        address        [1] Address,
        typeSpec       [2] TypeSpecification,
        data           [3] Data OPTIONAL,-- Field is only present if a
                        -- specific data value
                        -- is required for variable write tests. If not
                        -- present, it is assumed that any data value
                        -- consistent with the type specification is adequate.
        access         [4] VisibleString,     -- R, W, V
        kindOfVariable [5] VisibleString      -- Unnamed, SINGLE
      }
```

An example of the definitions follows.

```
addressVariableTableEntry {
index 0,
address {symbolicAddress "$A0:10"},
typeSpec {octet-string '256'},
      -- data not present
access "RW",
kindOfVariable "Unnamed"
}
```

#### C.1.5.5.3 Domain table

The list of supported Domains is defined via the following ASN.1 definitions. ASN.1 value notation shall be used for the actual definitions.

```
DomainTable ::= SEQUENCE OF DomainTableEntry

      DomainTableEntry ::= SEQUENCE {
        index              [0] INTEGER,
        domainName         [1] Identifier,
        listOfCapabilities [2] SEQUENCE OF VisibleString OPTIONAL,
        sharable           [3] BOOLEAN,
        deletable          [4] BOOLEAN,
        predefined         [5] BOOLEAN,
        loadDataFormat     CHOICE {
            formatOctetString [6] NULL,
            externalAbstractSyntax [7] SEQUENCE OF OBJECT IDENTIFIER
            }
      }
```

Examples of the definitions follow.

```
DomainTableEntry {
index 0,
domainName "Domain1",
listOfCapabilities {"capstring1","capstring2","capstring3"},
sharable TRUE,
deletable FALSE,
predefined TRUE,
loadDataFormat formatOctetString
}

DomainTableEntry {
index 1,
domainName "Domain2",
sharable TRUE,
deletable TRUE,
predefined FALSE,
loadDataFormat externalAbstractSyntax { { 1 0 9999 1992 1 3 } }
}

DomainTableEntry {
index 2,
domainName "Domain3",
listOfCapabilities {"overdraft","memprotect","phyIO"},
sharable FALSE,
deletable TRUE,
predefined TRUE,
loadDataFormat formatOctetString
}
```

```
DomainTableEntry {
index 3,
domainName "spc-gage",
listOfCapabilities {"feedback",
                    "wkl-fmt",
                    "offline-config",
                    "multidev"
                   },
sharable FALSE,
deletable TRUE,
predefined TRUE,
loadDataFormat externalAbstractSyntax { { 1 0 9999 1992 1 3 } }
}
```

#### C.1.5.5.4 Program Invocation table

The list of supported Program Invocations is defined via the following ASN.1 definitions. ASN.1 value notation shall be used for the actual definitions.

```
ProgramInvocationTable ::= SEQUENCE OF ProgramInvocationTableEntry

ProgramInvocationTableEntry ::= SEQUENCE {
  index             [0] INTEGER,
  pIname            [1] Identifier,
  listOfDomains     [2] SEQUENCE OF Identifier,
  reusable          [3] BOOLEAN,
  monitor           [4] BOOLEAN,
  deletable         [5] BOOLEAN,
  predefined        [6] BOOLEAN
  executionArgument CHOICE {
      stringArgument [7] VisibleString,
      externalAbstractSyntax [8] SEQUENCE OF OBJECT IDENTIFIER,
      } OPTIONAL
}
```

Examples of the definitions follow.

```
programInvocationTableEntry {
index 0,
pIname "Part-A",
listOfDomains {"Domain1","Domain2","Domain3"},
reusable TRUE,
monitor FALSE,
deletable TRUE,
predefined FALSE,
executionArgument {stringArgument "TheParameter"}
}
```

```
programInvocationTableEntry {
index 1,
pIname "Part-B",
listOfDomains {"spc-gage"}
reusable TRUE,
monitor FALSE,
deletable TRUE,
predefined TRUE
}

programInvocationTableEntry {
index 2,
pIname "Part-C",
listOfDomains {"Domain2"},
reusable FALSE,
monitor FALSE,
deletable FALSE,
predefined FALSE,
executionArgument externalAbstractSyntax { { 1 0 9999 1992 1 4 } }
}
```

### C.1.5.6 PICS/PIXIT Pro-forma

The following is a form that is actually filled out for an IUT.

**VendorName** _____

**ModelName** _____

**Revision** _____

List of Server services:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

List of Server parameterCBBs:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |

sNESTmin= sNESTmax=

sMaximumServicesOutstandingCallingMin=
sMaximumServicesOutstandingCallingMax=
sMaximumServicesOutstandingCalledMin=
sMaximumServicesOutstandingCalledMax=

sInitiateLocalDetailMin=
sInitiateLocalDetailMax=

List of Server Abstract Syntaxes

| |
|---|
| |
| |
| |
| |
| |

sLoadDataFormat ::=

List of Server Address formats:

| |
|---|
| |
| |

| |
|---|

List of Server data types:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

List of Client services:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

List of Client parameterCBBs:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |

cNESTmin= cNESTmax=

cMaximumServicesOutstandingCallingMin=
cMaximumServicesOutstandingCallingMax=
cMaximumServicesOutstandingCalledMin=
cMaximumServicesOutstandingCalledMax=

cInitiateLocalDetailMin=
cInitiateLocalDetailMax=

List of Client Abstract Syntaxes

| |
|---|
| |
| |
| |
| |
| |

cLoadDataFormat ::=

List of Client Address formats:

| |
|---|
| |
| |

| |
|---|

List of Client data types:

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

sAE-Title=
sPSAP=
sSSAP=
sTSAP=
sNSAP=
sMAC=
sLSAP=

cAE-Title=
cPSAP=
cSSAP=
cTSAP=
cNSAP=
cMAC=
cLSAP=

Named Variable Table:

Addressed Variable Table:

Domain Table:

Program Invocation Table:

## C.2 Basic functional tests

### C.2.1 Environment and general management test cases

#### C.2.1.1 EGM_INIT_01

Name: EGM_INIT_01

Purpose: Tests the ability to properly establish the MMS environment.

Inputs: Initiate parameters from the PICS

Initial Conditions: None

Sequence:

1.a The calling system sends:

```
Initiate-RequestPDU {
  localDetailCalling <LOCAL>,
  proposedMaxServOutstandingCalling <LOCAL>,
  proposedMaxServOutstandingCalled <LOCAL>,
  proposedDataStructureNestingLevel <LOCAL>,
  initRequestDetail{
    proposedVersionNumber 1,
    proposedParameterCBB <LOCAL>,
    servicesSupportedCalling <LOCAL>
  }
}
```

1.b The called system sends:

```
Initiate-ResponsePDU {
  localDetailCalled <LOCAL>,
  negotiatedMaxServOutstandingCalling <LOCAL>,
  negotiatedMaxServOutstandingCalled <LOCAL>,
  negotiatedDataStructureNestingLevel <LOCAL>,
  initResponseDetail{
    negotiatedVersionNumber 1,
    negotiatedParameterCBB <LOCAL>,
    servicesSupportedCalled <LOCAL>
  }
}
```

Pass Condition:

1. The values for localDetailCalling, proposedMaxServOutstandingCalling, proposedMaxServOutstandingCalled, proposedDataStructureNestingLevel, negotiatedParameterCBB and servicesSupportedCalling shall all be within the range specified in the requester PICS.

2. The values for localDetailCalled, negotiatedMaxServOutstandingCalling, negotiatedMaxServOutstandingCalled, negotiatedDataStructureNestingLevel and negotiatedParameterCBB shall be less than or equal to those proposed in 1.a and within the range specified in the responder PICS.

3. The value returned for servicesSupportedCalled shall be within the range specified in the responder PICS.

### C.2.1.2 EGM_CONC_01

Name: EGM_CONC_01

Purpose: Tests the ability to properly conclude the MMS environment.

Inputs: None

Initial Conditions:

1. A connection is established.

2. The server supports the server role for the conclude service.

Sequence:

1.a The client sends:

```
Conclude-RequestPDU
```

1.b The server sends:

```
Conclude-ResponsePDU
```

Pass Condition:

1. The MMS environment is terminated.

2. The client and server successfully release the underlying acse association.

#### C.2.1.3 EGM_ABRT_01

Name: EGM_ABRT_01

Purpose: Tests the ability of the client to abort the MMS environment.

Inputs: None

Initial Conditions:

1. A connection is established.

Sequence:

1.a The MMS application sends an acse abort.

Pass Condition:

1. The MMS environment is terminated.

2. The underlying acse association is aborted.

### C.2.2 VMD support test cases

The services included in this group are:

STAT - Status

USTA - Unsolicited Status

GNAM - Get Name List

IDEN - Identify

RNAM - Rename

GCAP - Get Capability List

### C.2.2.1 VMD_STAT_01

Name: VMD_STAT_01

Purpose: Tests the status service with extended derivation set to FALSE.

Inputs: None.

Initial Conditions:

1. A connection is established.

2. The server supports the server role for the status service.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  status FALSE
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  status {
    vmdLogicalStatus       <LOCAL>,
    vmdPhysicalStatus      <LOCAL>,
    localDetail            <LOCAL>
    -- Values of vmdLogicalStatus, vmdPhysicalStatus
    -- and localDetail are saved as <TMP:vmdLogicalStatus>,
    -- <TMP:vmdPhysicalStatus>. and <TMP:localDetail>.
    -- If localDetail is provided in part four
    -- of the server's PICS, it shall be sent
    -- as specified. If localDetail is not
    -- provided, it shall not be sent.
    }
  }
```

Pass Condition:

1. Values for <TMP:vmdLogicalStatus>, <TMP:vmdPhysicalStatus> and <TMP:localDetail> shall be valid.

### C.2.2.2 VMD_STAT_02

Name: VMD_STAT_02

Purpose: Tests the status service with extended derivation set to TRUE.

Inputs: If it is present, the optional procedure for evaluating status information will be extracted from the server PIXIT.

Initial Conditions:

1. A connection shall be established.

2. The server supports the server role for the status service.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  status TRUE
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  status {
    vmdLogicalStatus <LOCAL>,
    vmdPhysicalStatus      <LOCAL>,
    localDetail      <LOCAL>
    -- Values of vmdLogicalStatus, vmdPhysicalStatus and
    -- localDetail are saved as <TMP:vmdLogicalStatus>,
    -- <TMP:vmdPhysicalStatus> and <TMP:localDetail>.
    -- If localDetail is provided in part four of the server's PICS, then
    -- it shall be sent as specified. If localDetail is not provided, then
    -- this parameter shall not be sent.
    }
  }
```

Pass Condition:

1. Values for <TMP:vmdLogicalStatus> and <TMP:vmdPhysicalStatus> shall be valid.

### C.2.2.3 VMD_USTA_01

Name: VMD_USTA_01

Purpose: Tests normal reporting of the VMD unsolicited status.

Inputs: If it is present, the optional procedure for evaluating status information will be extracted from the server PIXIT.

Initial Conditions:

1. A connection shall be established.

2. The server supports the server role for the unsolicitedStatus service.

Sequence:

1.a The server sends:

```
unconfirmed-PDU {
  unsolicitedStatus {
    vmdLogicalStatus <LOCAL>,
    vmdPhysicalStatus      <LOCAL>,
    localDetail      <LOCAL>
    -- Values of vmdLogicalStatus, vmdPhysicalStatus and
    -- localDetail are saved as <TMP:vmdLogicalStatus>,
    -- <TMP:vmdPhysicalStatus> and <TMP:localDetail>.
    -- If localDetail is provided in part four of the server's PICS, then
    -- it shall be sent as specified. If localDetail is not provided, then
    -- this parameter shall not be sent.
    }
  }
```

Pass Condition:

1. Values for <TMP:vmdLogicalStatus> and <TMP:vmdPhysicalStatus> shall be valid.

### C.2.2.4 VMD_GNAM_01

Name: VMD_GNAM_01

Purpose: Tests the retrieval of named variables existing in the VMD.

Inputs: A list of variable names expected to be returned. This includes any permanent named variables, named variables created by local means and named variables created by MMS services.

Initial Conditions:

1. A connection is established.

2. The server supports the server role for the getNameList service.

Sequence:

A temporary variable of type Identifier, <TMP:continueAfterName>, is initialized to be "".

A temporary variable of type SEQUENCE OF Identifier, <TMP:listOfIdentifier>, is initialized to be {}.

1. The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getNameList {
    extendedObjectClass objectClass namedVariable,
    objectScope vmdSpecific NULL
    }
  }
```

There shall be zero or more occurrences of subsequence two.

2.a The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getNameList {
    listOfIdentifier <LOCAL>,
    -- Values of listOfIdentifier are concatenated with the
    -- contents of temporary variable <TMP:listOfIdentifier>.
    -- <TMP:continueAfterName> is set equal to the last element
    -- in the sequence of listOfIdentifier.
    moreFollows TRUE
    }
  }
```

2.b The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getNameList {
    extendedObjectClass objectClass namedVariable,
    objectScope vmdSpecific NULL,
    continueAfter <TMP:continueAfterName>
    }
  }
```

3. The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getNameList {
    listOfIdentifier <LOCAL>,
    -- Values of listOfIdentifier are concatenated with the
    -- contents of temporary variable <TMP:listOfIdentifier>.
    -- <TMP:continueAfterName> is set equal to the last element
    -- in the sequence of listOfIdentifier.
    moreFollows FALSE
    }
  }
```

Pass Condition:

1. The sequence of values contained in <TMP:listOfIdentifier> after parsing is complete shall match the expected sequence of identifiers.

### C.2.2.5 VMD_IDEN_01

Name: VMD_IDEN_01

Purpose: Tests the retrieval of identify information.

Inputs: No additional input information is required.

Initial Conditions:

1. A connection is established.

2. The server supports the server role for the identify service.

Sequence:

1. The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  identify NULL
  }
```

2. The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  identify {
    vendorName <sPICS:PartOne:vendorName>,
    modelName  <sPICS:PartOne:modelName>,
    revision         <sPICS:PartOne:revision>
    listOfAbstractSyntaxes <LOCAL>
    -- The listOfAbstractSyntaxes is saved as temporary variable
    -- <TMP:listOfAbstractSyntaxes>.
    }
  }
```

Pass Condition:

1. The content of <TMP:listOfAbstractSyntaxes> shall be the union of the lists of abstract syntaxes from <sPICS:PartOne:CSAbstractSyntaxes>, <sPICS:PartFour:loadData> and <sPICS:PartFour:executionArgument>.

### C.2.2.6 Rename test cases

### C.2.2.7 VMD_GCAP_01

Name: VMD_GCAP_01

Purpose: Tests the retrieval of vmd capabilities.

Inputs: The vmd capabilities are extracted from the server PICS.

Initial Conditions:

1. A connection shall be established.

2. The server supports the server role for the getCapabilityList service.

Sequence:

A temporary variable of type SEQUENCE OF VisibleString, <TMP:capabilities>, is initialized to be {}.

1. The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getCapabilityList {
    }
  }
```

There shall be zero or more occurrences of subsequence two.

2.a The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getCapabilityList {
    listOfCapabilities <LOCAL>,
    -- Values of listOfCapabilities are concatenated with the
    -- contents of temporary variable <TMP:capabilities>.
    -- <TMP:continueAfterName> is set equal to the last element
    -- in the sequence of listOfCapabilities.
    moreFollows TRUE
    }
  }
```

2.b The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getCapabilityList {
    continueAfter <TMP:continueAfterName>
    -- The continueAfter parameter is not sent if its contents are
    -- the NULL string
    }
  }
```

3. The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getCapabilityList {
    listOfCapabilities <LOCAL>,
    -- Values of listOfCapabilities are concatenated with the
    -- contents of temporary variable <TMP:capabilities>.
    -- <TMP:continueAfterName> is set equal to the last element
    -- in the sequence of listOfCapabilities.
    moreFollows FALSE
    }
  }
```

Pass Condition:

1. The sequence of values contained in <TMP:capabilities> after parsing is complete shall match the sequence of values contained in <sPICS:capabilitiesOfVmd>.

## C.2.3 Domain management test cases

### C.2.3.1 DOM_UPLD_01

Name: DOM_UPLD_01

Purpose: Tests an upload of a single Domain.

Inputs: 1.X - integer index of the required Domain in the PIXIT list of Domains.

Initial Conditions:

1. A connection is established with InitiateUploadSequence, UploadSegment, and TerminateUploadSequence supported by the server.

2. Domain named in <PIXIT:DOM.X.DomainName> exists in the server in a state that can be that it can be uploaded.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  initiateUploadSequence
      <sPIXIT:DOM:X:DomainName>
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  initiateUploadSequence {
    ulsmID <LOCAL>,
    -- <TMP:ulsmID> is set equal to ulsmID.
    listOfCapabilities <sPIXIT:DOM:X:listOfCapabilities>
  }
}
```

2. The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  uploadSegment <TMP:ulsmID>
}
```

There shall be zero or more occurrences of subsequence three.

3.a The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  uploadSegment {
    loadData <LOCAL>,
    moreFollows TRUE
  }
}
```

3.b The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  uploadSegment <TMP:ulsmID>
}
```

4. The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  uploadSegment {
    loadData <LOCAL>,
    moreFollows FALSE
  }
}
```

5.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  terminateUploadSequence
       <TMP:ulsmID>
}
```

5.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  terminateUploadSequence NULL
}
```

Pass Condition:

1. The sequence of received segments contains sufficient information to reconstruct the original Domain content.

### C.2.3.2 DOM_DNLD_01

Name: DOM_DNLD_01

Purpose: Tests the ability to download a Domain into the server system.

Inputs: 1. X - Integer value indicating the index in the list of Domains in the server PIXIT.

Initial Conditions:

1. A connection is established with InitiateDownloadSequence supported by the server and DownloadSegment and TerminateDownloadSequence supported by the client.

2. Domain named in <sPIXIT:DOM:X:DomainName> does not exist in the server.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  initiateDownloadSequence {
    DomainName <sPIXIT:DOM:X:DomainName>,
    listOfCapabilities
     <sPIXIT:DOM:X:listOfCapabilities>,
    sharable <sPIXIT:DOM:X:sharable>
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  initateDownloadSequence NULL
}
```

2. The server sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  downloadSegment <sPIXIT:DOM:X:DomainName>
}
```

There shall be exactly one occurrence of subsequence three.

3.a. The client sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  downloadSegment {
    loadData <LOCAL>,
    moreFollows TRUE
    }
}
```

3.b. The server sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  downloadSegment <sPIXIT:DOM:X:DomainName>
}
```

4. The client sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  downloadSegment {
    loadData <LOCAL>,
    moreFollows FALSE
  }
}
```

4.a The server sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  terminateDownloadSequence{
    DomainName <sPIXIT:DOM:X:DomainName>
    -- discard shall not be present
  }
}
```

4.b The client sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  terminateDownloadSequence NULL
}
```

Pass Condition:

1. A Domain exists on the server with name <sPIXIT:DOM:X:DomainName> and is in the Ready state.

### C.2.3.3 DOM_DELE_01

Name: DOM_DELE_01

Purpose: Test the ability to delete a Domain on the server.

Inputs: 1. X - Integer value indicating the index of the Domain to be deleted in the server PIXIT.

Initial Conditions:

1. A connection is established with DeleteDomain supported by the server.

2. Domain named in <sPIXIT:DOM:X:DomainName> exists, is in the READY state and has an MMS deletable attribute set to TRUE.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  deleteDomain <sPIXIT:DOM:X:DomainName>
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  deleteDomain NULL
}
```

Pass Condition:

1. Domain <sPIXIT:DOM:X:DomainName> no longer present on the server device.

### C.2.3.4 DOM_GETA_01

Name: DOM_GETA_01

Purpose: Test the ability to get the attributes of a Domain on the server.

Inputs: 1. X - Integer value indicating the index of the Domain to be deleted in the server PIXIT.

Initial Conditions:

1. A connection is established with GetDomainAttributes supported by the server.

2. Domain named in <sPIXIT:DOM:X:DomainName> exists.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getDomainAttributes <sPIXIT:DOM:X:DomainName>
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getDomainAttributes {
    listOfCapabilities <sPIXIT:DOM:X:listOfCapabilities>,
    state <LOCAL>,
    mmsDeletable <sPIXIT:DOM:X:mmsDeletable>,
    sharable <sPIXIT:DOM:X:sharable>,
    listOfProgramInvocations <LOCAL>,
    uploadInProgress <LOCAL>
  }
}
```

Pass Condition:

## C.2.4 Program Invocation test cases

### C.2.4.1 PIM_CREA_01

Name: PIM_CREA_01

Purpose: Test the ability to create a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be created in the server PIXIT.

Initial Conditions:

1. A connection is established with CreateProgramInvocation supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> does not exist in the server.

3. Domains named in <sPIXIT:PI:X:listOfDomainNames> exist and are able to be incorporated into a Program Invocation.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  CreateProgramInvocation {
    programInvocationName <sPIXIT:PI:X:PIName>,
    listOfDomainNames
        <sPIXIT:PI:X:listOfDomainNames>,
    reusable <sPIXIT:PI:X:reusable>,
    monitorType <sPIXIT:PI:X:monitorType>
    -- The monitorType parameter is not sent if
    -- the server PIXIT indicates the Program Invocation is not
    -- monitored.
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  createProgramInvocation NULL
}
```

Pass Condition:

1. A Program Invocation exists on the server with name <sPIXIT:PI:X:PIName> and is in the IDLE state.

### C.2.4.2 PIM_DELE_01

Name: PIM_DELE_01

Purpose: Test the ability to delete a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be deleted in the server PIXIT.

Initial Conditions:

1. A connection is established with DeleteProgramInvocation supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server.

3. The Program Invocation is in one of the following states - IDLE, STOPPED or UNRUNNABLE.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  DeleteProgramInvocation
       <sPIXIT:PI:X:PIName>
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  deleteProgramInvocation NULL
}
```

Pass Condition:

1. The Program Invocation is deleted.

### C.2.4.3 PIM_STRT_01

Name: PIM_STRT_01

Purpose: Test the ability to start a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be started in the server PIXIT.

Initial Conditions:

1. A connection is established with Start supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server and be in the IDLE state.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  start {
    programInvocationName <sPIXIT:PI:X:PIName>,
    executionArgument <sPIXIT:PI:X:executionArgument>
    -- This parameter is not sent if the server
    -- PIXIT does not specify an execution argument.
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  start NULL
}
```

Pass Condition:

1. The Program Invocation makes the transition to the RUNNING state.

### C.2.4.4 PIM_STOP_01

Name: PIM_STOP_01

Purpose: Test the ability to stop a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be stopped in the server PIXIT.

Initial Conditions:

1. A connection is established with Stop supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server and be in the RUNNING state.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  stop {
    programInvocationName <sPIXIT:PI:X:PIName>
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  stop NULL
}
```

Pass Condition:

1. The Program Invocation makes the transition to the STOPPED state.

### C.2.4.5 PIM_RESM_01

Name: PIM_RESM_01

Purpose: Test the ability to resume a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be resumed in the server PIXIT.

Initial Conditions:

1. A connection is established with Resume supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server and be in the STOPPED state.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  resume {
    programInvocationName <sPIXIT:PI:X:PIName>,
    executionArgument <sPIXIT:PI:X:executionArgument>
    -- This parameter is not sent if the server
    -- PIXIT does not specify an execution argument.
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  resume NULL
}
```

Pass Condition:

1. The Program Invocation makes the transition to the RUNNING state.

### C.2.4.6 PIM_REST_01

Name: PIM_REST_01

Purpose: Test the ability to reset a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Inovocation to be reset in the server PIXIT.

Initial Conditions:

1. A connection is established with Reset supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server and is in the STOPPED state.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  reset {
    programInvocationName <sPIXIT:PI:X:PIName>
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  reset NULL
}
```

Pass Condition:

1. The Program Invocation makes the transition to the IDLE state.

### C.2.4.7 PIM_GPIA_01

Name: PIM_GPIA_01

Purpose: Test the ability to get the attributes of a Program Invocation on the server.

Inputs: 1. X - Integer value indicating the index of the Program Invocation to be reset in the server PIXIT.

Initial Conditions:

1. A connection shall be established with GetProgramInvocationAttributes supported by the server.

2. Program Invocation named in <sPIXIT:PI:X:PIName> exists on the server.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeID <ID>,
  getProgramInvocationAttributes
    <sPIXIT:PI:X:PIName>
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeID <ID>,
  getProgramInvocationAttributes {
    state <LOCAL>,
    listOfDomainNames <sPIXIT:PI:X:listOfDomainNames>,
    mmsDeletable <sPIXIT:PI:X:mmsDeletable>,
    reusable <sPIXIT:PI:X:reusable>,
    monitor <sPIXIT:PI:X:monitor>,
    executionArgument <sPIXIT:PI:X:executionArgument>
  }
}
```

Pass Condition:

## C.2.5 Variable access test cases

### C.2.5.1 VAR_READ_01

Name: VAR_READ_01

Purpose: To test the Read service for a single NAMED variable marked as readable (R,RW,RWV) in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. Connection established with Read negotiated by the server.

2. VNAM shall be negotiated on the connection.

3. Appropriate values for STR1, STR2, and NEST shall be negotiated to support variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification name
        <sPIXIT:NAMED_VAR:X:Name>
    }
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success>.
    }
  }
}
```

Pass Condition:

<TMP:success> is consistent with the type specification in <sPIXIT:NAMED_VAR:X:TypeSpec>.

### C.2.5.2 VAR_READ_02

Name: VAR_READ_02

Purpose: To test the Read service for a single UNNAMED variable marked as readable (R,RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Read supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
    }
  }
}
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
        -- The value of success is saved as <TMP:success>.
    }
  }
}
```

Pass Condition:

<TMP:success> is consistent with the type specification in <sPIXIT:ADDR_VAR:X:TypeSpec>.

### C.2.5.3 VAR_READ_03

Name: VAR_READ_03

Purpose: To test the Read service for a single SINGLE variable marked as readable (R,RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Read supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a. Client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
      }
    }
  }
}
```

1.b. Server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success>.
    }
  }
}
```

Pass Condition:

<TMP:success> is consistent with the type specification in <sPIXIT:ADDR_VAR:X:TypeSpec>.

### C.2.5.4 VAR_WRIT_01

Name: VAR_WRIT_01

Inputs: To test the Write service for a single NAMED variable marked as writable (RW,RWV) in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. A connection is established with Write supported by the server.

2. VNAM is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated as appropriate to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification name
       <sPIXIT:NAMED_VAR:X:Name>
      },
      listOfData {
        <sPIXIT:NAMED_VAR:X:Data>
        }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

Pass Condition:

### C.2.5.5 VAR_WRIT_02

Name: VAR_WRIT_02

Purpose: To test the Write service for a single UNNAMED variable marked as writable (RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Write supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
      },
    listOfData {
      <sPIXIT:ADDR_VAR:X:Data>
      }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

Pass Condition:

### C.2.5.6 VAR_WRIT_03

Name: VAR_WRIT_03

Purpose: To test the Write service for a single SINGLE variable marked as writable (RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Write supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
      },
    listOfData {
      <sPIXIT:ADDR_VAR:X:Data>
      }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

Pass Condition:

### C.2.5.7 VAR_RMWV_01

Name: VAR_RMWV_01

Purpose: To test the ability to read, modify, write, and verify the changed value of a single NAMED variable marked as read-write-verifiable (RWV) in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. A connection is established with Read supported by the server.

2. VNAM is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification name
        <sPIXIT:NAMED_VAR:X:Name>
      }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success1>.
      }
    }
  }
```

2. The client alters <TMP:success1>. The new value shall be consistent with the type specification <sPIXIT:NAMED_VAR:X:TypeSpec>.

3.a The client sends:

```
confirmed-RequestPDU {
  invokeId < ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification name
        <sPIXIT:NAMED_VAR:X:Name>
      },
    listOfData {
      <TMP:success1>
      }
    }
  }
```

3.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

4.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification name
        <sPIXIT:NAMED_VAR:X:Name>
      }
    }
  }
```

4.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success2>.
      }
    }
  }
```

Pass Condition: <TMP:success1> shall be equal to <TMP:success2>.

### C.2.5.8 VAR_RMWV_02

Name: VAR_RMWV_02

Purpose: To test the ability to read, modify, write, and verify the changed value of a single UNNAMED variable marked as read-write-verifiable (RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Read supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
      }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success1>.
      }
    }
  }
```

2. The client alters <TMP:success1>. The new value shall be consistent with the type specification <sPIXIT:ADDR_VAR:X:TypeSpec>.

3.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
      },
    listOfData {
      <TMP:success1>
      }
    }
  }
```

3.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

4.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
      }
    }
  }
```

4.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success2>.
      }
    }
  }
```

Pass Condition: <TMP:success1> shall be equal to <TMP:success2>.

### C.2.5.9 VAR_RMWV_03

Name: VAR_RMWV_03

Purpose: To test the ability to read, modify, write, and verify the changed value of a single SINGLE variable marked as read-write-verifiable (RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with Read supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
      }
    }
  }
```

1.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success1>.
      }
    }
  }
```

2. The client alters <TMP:success1>. The new value shall be consistent with the type specification <sPIXIT:ADDR_VAR:X:TypeSpec>.

3.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  write {
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
      },
    listOfData {
      <TMP:success1>
      }
    }
  }
```

3.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  write {
    success NULL
    }
  }
```

4.a The client sends:

```
confirmed-RequestPDU {
  invokeId <ID>,
  read {
    specificationWithResult FALSE,
    -- As an option, specificationWithResult is omitted.
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
      }
    }
  }
```

4.b The server sends:

```
confirmed-ResponsePDU {
  invokeId <ID>,
  read {
    -- variableAccessSpecification is not present
    listOfAccessResult {
      success <LOCAL>
      -- The value of success is saved as <TMP:success2>.
      }
    }
  }
```

Pass Condition: <TMP:success1> shall be equal to <TMP:success2>.

### C.2.5.10 VAR_IRPT_01

Name: VAR_IRPT_01

Purpose: To test the InformationReport service for a single NAMED variable marked as readable (R,RW,RWV) in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. A connection is established with InformationReport supported by the client.

2. VNAM is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The server sends:

```
unconfirmed-PDU {
  informationReport {
    variableAccessSpecification listOfVariable {
      variableSpecification name
        <sPIXIT:NAMED_VAR:X:Name>
      },
    listOfAccessResult {
      success <LOCAL>
      -- the value of success is saved as <TMP:success>.
      }
    }
  }
```

Pass Condition: <TMP:success> is consistent with the type specification in <sPIXIT:NAMED_VAR:X:TypeSpec>.

### C.2.5.11 VAR_IRPT_02

Name: VAR_IRPT_02

Purpose: To test the InformationReport service for a single UNNAMED variable marked as readable (R,RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with InformationReport supported by the client.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The server sends:

```
unconfirmed-PDU {
  informationReport {
    variableAccessSpecification listOfVariable {
      variableSpecification address
        <sPIXIT:ADDR_VAR:X:Address>
      },
    listOfAccessResult {
      success <LOCAL>
      -- the value of success is saved as <TMP:success>.
      }
    }
  }
```

Pass Condition: <TMP:success> is consistent with the type specification in <sPIXIT:ADDR_VAR:X:TypeSpec>.

### C.2.5.12 VAR_IRPT_03

Name: VAR_IRPT_03

Purpose: To test the InformationReport service for a single SINGLE variable marked as readable (R,RW,RWV) in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the server PIXIT ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with InformationReport supported by the client.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The server sends:

```
unconfirmed-PDU {
  informationReport {
    variableAccessSpecification listOfVariable {
      variableSpecification variableDescription {
        address <sPIXIT:ADDR_VAR:X:Address>,
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
      },
    listOfAccessResult {
      success <LOCAL>
      -- the value of success is saved as <TMP:success>.
      }
    }
  }
```

Pass Condition: <TMP:success> is consistent with the type specification in <sPIXIT:ADDR_VAR:X:TypeSpec>.

### C.2.5.13 VAR_GVAA_01

Name: VAR_GVAA_01

Purpose: To test the GetVariableAccessAttributes service for a single NAMED variable with NON PUBLIC access method as specified in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. A connection is established with GetVariableAccessAttributes supported by the server.

2. VNAM is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        name <sPIXIT:NAMED_VAR:X:Name>
        }
    }
```

1.b The server sends:

```
confirmed-ResponsePDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        mms_deletable <sPIXIT:NAMED_VAR:X:Deletable>,
        -- the address parameter is not present
        typeSpecification <sPIXIT:NAMED_VAR:X:TypeSpec>
        }
    }
```

Pass Condition:

### C.2.5.14 VAR_GVAA_02

Name: VAR_GVAA_02

Purpose: To test the GetVariableAccessAttributes service for a single NAMED variable with PUBLIC access method as specified in the server PIXIT NAMED_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:NAMED_VAR table of the server.

Initial Conditions:

1. A connection is established with GetVariableAccessAttributes supported by the server.

2. VNAM is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        name <sPIXIT:NAMED_VAR:X:Name>
        }
    }
```

1.b The server sends:

```
confirmed-ResponsePDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        mms_deletable <sPIXIT:NAMED_VAR:X:Deletable>,
        address <sPIXIT:NAMED_ADDR:X:Address>
        typeSpecification <sPIXIT:NAMED_VAR:X:TypeSpec>
        }
    }
```

Pass Condition:

### C.2.5.15 VAR_GVAA_03

Name: VAR_GVAA_03

Purpose: To test the GetVariableAccessAttributes service for a single UNNAMED variable with as specified in the server PIXIT ADDR_VAR table.

Inputs: X - integer value indicating the INDEX into the sPIXIT:ADDR_VAR table of the server.

Initial Conditions:

1. A connection is established with GetVariableAccessAttributes supported by the server.

2. VADR is supported on the connection.

3. Appropriate values for STR1, STR2, and NEST are negotiated to support the variable being tested.

Sequence:

1.a The client sends:

```
confirmed-RequestPDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        address <sPIXIT:ADDR_VAR:X:Address>
        }
    }
```

1.b The server sends:

```
confirmed-ResponsePDU {
    invokeId <ID>,
    getVariableAccessAttributes {
        mms_deletable <sPIXIT:ADDR_VAR:X:Deletable>,
        -- the address parameter is not present
        typeSpecification <sPIXIT:ADDR_VAR:X:TypeSpec>
        }
    }
```

Pass Condition:

### C.2.6 Semaphore management test cases

### C.2.7 Operator communication test cases

### C.2.8 Event management test cases

### C.2.9 Journal management test cases

## C.3 Basic functional test script language

## C.4 References