NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
1334-4-41

Première édition
First edition
1996-07

Automatisation de la distribution à l'aide de systèmes de communication à courants porteurs –

Partie 4:
Protocoles de communication de données –
Section 41: Protocoles d'application –
Spécification des messages de ligne de distribution

Distribution automation using distribution line carrier systems –

Part 4:
Data communication protocols –
Section 41: Application protocols –
Distribution line message specification

Numéro de référence
Reference number
CEI/IEC 1334-4-41: 1996

Numéros des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000.

Publications consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Validité de la présente publication

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique.

Des renseignements relatifs à la date de reconfirmation de la publication sont disponibles dans le Catalogue de la CEI.

Les renseignements relatifs à des questions à l'étude et des travaux en cours entrepris par le comité technique qui a établi cette publication, ainsi que la liste des publications établies, se trouvent dans les documents ci-dessous:

- **«Site web» de la CEI***
- **Catalogue des publications de la CEI**
  Publié annuellement et mis à jour régulièrement
  (Catalogue en ligne)*
- **Bulletin de la CEI**
  Disponible à la fois au «site web» de la CEI* et comme périodique imprimé

Terminologie, symboles graphiques et littéraux

En ce qui concerne la terminologie générale, le lecteur se reportera à la CEI 60050: *Vocabulaire Electrotechnique International* (VEI).

Pour les symboles graphiques, les symboles littéraux et les signes d'usage général approuvés par la CEI, le lecteur consultera la CEI 60027: *Symboles littéraux à utiliser en électrotechnique,* la CEI 60417: S*ymboles graphiques utilisables sur le matériel. Index, relevé et compilation des feuilles individuelles,* et la CEI 60617: *Symboles graphiques pour schémas.*

* Voir adresse «site web» sur la page de titre.

Numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series.

Consolidated publications

Consolidated versions of some IEC publications including amendments are available. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Validity of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information relating to the date of the reconfirmation of the publication is available in the IEC catalogue.

Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is to be found at the following IEC sources:

- **IEC web site***
- **Catalogue of IEC publications**
  Published yearly with regular updates
  (On-line catalogue)*
- **IEC Bulletin**
  Available both at the IEC web site* and as a printed periodical

Terminology, graphical and letter symbols

For general terminology, readers are referred to IEC 60050: *International Electrotechnical Vocabulary* (IEV).

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to publications IEC 60027: *Letter symbols to be used in electrical technology*, IEC 60417: *Graphical symbols for use on equipment. Index, survey and compilation of the single sheets* and IEC 60617: *Graphical symbols for diagrams.*

* See web site address on title page.

NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
1334-4-41

Première édition
First edition
1996-07

Automatisation de la distribution à l'aide de systèmes de communication à courants porteurs –

Partie 4: Protocoles de communication de données – Section 41: Protocoles d'application – Spécification des messages de ligne de distribution

Distribution automation using distribution line carrier systems –

Part 4: Data communication protocols – Section 41: Application protocols – Distribution line message specification

Bureau central de la Commission Electrotechnique Internationale 3, rue de Varembé Genève Suisse

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE XF

Pour prix, voir catalogue en vigueur
For price, see current catalogue

SOMMAIRE

CONTENTS

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

AUTOMATISATION DE LA DISTRIBUTION À L'AIDE DE SYSTÈMES DE COMMUNICATION À COURANTS PORTEURS –

Partie 4: Protocoles de communication de données – Section 41: Protocoles d'application – Spécification des messages de ligne de distribution

AVANT-PROPOS

1) La CEI (Commission Electrotechnique Internationale) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI, entre autres activités, publie des Normes internationales. Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des condition fixées par accord entre les deux organisations.

2) Les décisions ou accords officiels de la CEI en ce qui concerne les questions techniques, préparés par des comités d'études où sont représentés tous les Comités nationaux s'intéressant à ces questions, expriment dans la plus grande mesure possible un accord international sur les sujets examinés.

3) Ces décisions constituent des recommandations internationales publiées sous forme de normes, de rapports techniques ou de guides et agréées comme telles par les Comités nationaux.

4) Dans le but d'encourager l'unification internationale, les Comités nationaux de la CEI s'engagent à appliquer de façon transparente, dans toute la mesure du possible, les Normes internationales de la CEI dans leurs normes nationales et régionales. Toute divergence entre la norme de la CEI et la norme nationale ou régionale correspondante doit être indiquée en termes clairs dans cette dernière.

5) L'attention est attirée sur le fait que certains des éléments de la présente Norme internationale peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

6) La CEI n'a fixé aucune procédure concernant le marquage comme indication d'approbation et sa responsabilité n'est pas engagée quand il est déclaré qu'un matériel est conforme à l'une de ses normes.

La Norme internationale CEI 1334-4-41 a été établie par le comité d'études 57 de la CEI: Conduite des systèmes de puissance et communications associées.

Le texte de la présente norme est issu des documents suivants:

FDIS	Rapport de vote
57/261/FDIS	57/284/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

L'annexe A fait partie intégrante de cette norme.

L'annexe B est donnée uniquement à titre d'information.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

DISTRIBUTION AUTOMATION USING DISTRIBUTION LINE CARRIER SYSTEMS –

Part 4 : Data communication protocols – Section 41: Application protocols – Distribution line message specification

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international cooperation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of the IEC on technical matters express as nearly as possible an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.

3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical reports or guides and they are accepted by the National Committees in that sense.

4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.

5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.

6) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 1334-4-41 has been prepared by IEC technical committee 57: Power system control and associated communications.

The text of this standard is based on the following documents:

FDIS	Report on voting
57/261/FDIS	57/284/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

Annex A forms an integral part of this standard.

Annex B is for information only.

INTRODUCTION

La présente spécification définit la spécification des messages de ligne de distribution (DLMS) au sein de la couche application de l'OSI selon les modalités ci-dessous :

a) modèle abstrait définissant les interactions entre les utilisateurs des services ;

b) fonctionnalités, visibles de l'extérieur, des implémentations conformes à cette spécification, sous la forme d'exigences de procédures associées à l'exécution de demandes de services ;

c) actions primitives et événements des services ;

d) paramètre "data" (données) associé à chaque action primitive et à chaque événement ;

e) relation entre, et les séquences valides de, ces actions et de ces événements.

Les services définis dans la présente spécification sont fournis par le protocole DLMS. Ils peuvent être utilisés par d'autres éléments de service de la couche application ou par d'autres éléments de processus d'application.

La présente spécification ne décrit pas d'implémentations particulières ni de produits et pas davantage les contraintes d'implémentation des entités et des interfaces dans un système informatique. La présente spécification définit les fonctionnalités visibles de l'extérieur des implémentations ainsi que les exigences de conformité pour de telles fonctionnalités.

INTRODUCTION

This specification defines the distribution line message specification (DLMS) within the OSI application layer in terms of:

a) an abstract model defining the interaction between users of services;

b) the externally visible functionality of implementations conforming to this specification, in the form of procedural requirements associated with the execution of service requests;

c) the primitive actions and events of services;

d) the parameter data associated with each primitive actions and events;

e) the relationship between, and the valid sequences of, these actions and events.

The services defined in this specification are provided by the distribution line message specification protocol. They may be used by other application layer service elements or by other elements of the application process.

This specification does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces within a computer system. This specification specifies the externally visible functionality of implementations and the conformance requirements for such functionality.

AUTOMATISATION DE LA DISTRIBUTION À L'AIDE DE SYSTÈMES DE COMMUNICATION À COURANTS PORTEURS –

Partie 4: Protocoles de communication de données –
Section 41: Protocoles d'application – Spécification des messages de ligne de distribution

1 Domaine d'application

Le domaine d'application des spécifications fournies dans les sections de la partie 4 est la communication par la technique dite des courants porteurs (DLC), à la fois sur les réseaux basse et moyenne tensions. La palette d'applications servies par ce procédé de communication est étendue et ne saurait être décrite de façon exhaustive dans la présente section; on peut citer à titre d'exemple le contrôle et la surveillance de réseaux de distribution, la diffusion de commandes, le contrôle des interfaces clientèles, de l'éclairage public, la supervision des feux de signalisation routière, le relevé automatique de compteurs, etc.

Des extensions à d'autres supports de communication sont également autorisées.

Distribution Line Message Specification (DLMS - Spécification de messages de lignes de distribution) est la spécification d'une couche application destinée à supporter des échanges de messages entre équipements de distribution dans un environnement informatisé. Dans la présente spécification, cet environnement est appelé "environnement de distribution". Cette spécification ne définit pas un jeu complet de services pour la programmation à distance d'équipements.

DISTRIBUTION AUTOMATION USING DISTRIBUTION LINE CARRIER SYSTEMS –

Part 4 : Data communication protocols – Section 41: Application protocols – Distribution line message specification

1 Scope

The scope of application of the specifications of the sections of part 4 is the communication through the so-called distribution line carrier technology (DLC) on both low and medium voltage distribution network. The application range based on telecommunication processes is wide and cannot be described exhaustively in this section; application examples are: control and monitoring of the distribution network, broadcasting of orders, control of user interfaces, public lighting, traffic lights supervision, automatic meter reading, etc.

Extensions to other communication media are also allowed.

The distribution line message specification (DLMS) is an application layer specification designed to support messaging communications to and from distribution devices in a computer integrated environment. This environment is referred in this specification as the distribution environment. This specification does not specify a complete set of services for remote programming of devices.

2 Références normatives

Les documents normatifs suivants contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente section de la CEI 1334-4. Au moment de la publication, les éditions indiquées étaient en vigueur. Tout document normatif est sujet à révision et les parties prenantes aux accords fondés sur la présente section de la CEI 1334-4 sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des documents normatifs indiqués ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur.

CEI/FDIS 1334-4-42: *Automatisation de la distribution à l'aide de systèmes de communication à courants porteurs – Partie 4: Protocoles de communication de données – Section 42: Protocoles d'application – Couche application* [1]

ISO 7498 : 1984, *Systèmes de traitement de l'information - Interconnexion des systèmes ouverts - Modèle de référence de base*

ISO/IEC 7498-3 : 1989, *Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Modèle de référence de base - Partie 3 : Dénomination et adressage*

ISO/IEC/TR 8509 : 1987, *Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Conventions de service*

ISO/IEC 8649 : 1988, *Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Définition du service pour l'élément du service de contrôle d'association*

ISO/IEC 8650 : 1988, *Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Spécification du protocole pour l'élément de service de contrôle d'association*

ISO/IEC 8824 : 1990, *Technologies de l'information - Interconnexion de systèmes ouverts - Spécification de la notation de syntaxe abstraite numéro 1 (ASN.1)* (publiée actuellement en anglais seulement)

ISO/IEC 8825 : 1990, *Technologies de l'information - Interconnexion de systèmes ouverts - Spécification de règles de base pour coder la notation de syntaxe abstraite numéro une (ASN.1*) (publiée actuellement en anglais seulement)

ISO/IEC 9545 : 1994, *Technologies de l'information - Interconnexion de systèmes ouverts - Structure de la couche application* (publiée actuellement en anglais seulement)

[1] Actuellement au stade de projet final de Norme internationale (57/265/FDIS).

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this section of IEC 1334-4. At the time of publication, the editions indicated were valid. All normative documents are subject to revision, and parties to agreements based on this section of IEC 1334-4 are encouraged to investigate the possibility of applying the most recent editions of the normative documents listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

IEC/FDIS 1334-4-42*: Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 42: Application protocols – Application layer* [1]

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

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

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

ISO/IEC 8649: 1988, *Information processing systems - Open Systems Interconnection - Service definition for the Association Control Service Element*

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

ISO/IEC 8824: 1990, *Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825: 1990, *Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*

ISO/IEC 9545: 1994, *Information technology - Open Systems Interconnection - Application Layer structure*

[1] At present at the stage of Final Draft International Standard (57/265/FDIS).

3 Définitions

3.1 Définitions du modèle de référence

La présente spécification repose sur des concepts développés dans le modèle de référence de base pour l'interconnexion de systèmes ouverts (ISO 7498).

Les termes ci-dessous, définis dans cette Norme internationale, y sont employés.

a) **AE : entité d'application ;**

b) **AP : processus d'application ;**

c) **ASE : élément de service d'application ;**

d) **système ouvert ;**

e) **(N)-protocol : protocole (N) ;**

f) **(N)-protocol-data-unit (N-PDU) : unité de données du protocole (N) ;**

g) **(N)-service-access-point (N-SAP) : point d'accès au service (N) ;**

h) **(N)-layer : couche (N) ;**

i) **système ;**

j) **(N)-user-data : données utilisateur (N).**

3.2 Définitions des conventions de service

La présente spécification utilise, quand ils sont applicables à DLMS, les termes ci-dessous définis dans l'ISO/IEC/TR 8509 :

a) **primitive** (*primitive)*;

b) **request** *(demande)*;

c) **indication** *(indication)*;

d) **response** *(réponse)*;

e) **confirm** *(confirmation)*;

3 Definitions

3.1 Reference model definitions

This specification is based on the concepts developed in the Basic Reference Model for Open Systems Interconnection (ISO 7498).

It makes use of the following terms defined in that International Standard:

a) **application-entity (AE);**

b) **application-process (AP);**

c) **application service element (ASE);**

d) **open system;**

e) **(N)-protocol;**

f) **(N)-protocol-data-unit (N-PDU);**

g) **(N)-service-access-point (N-SAP);**

h) **(N)-layer;**

i) **system;**

j) **(N)-user-data.**

3.2 Service convention definitions

This specification makes use of the following terms defined in ISO/IEC/TR 8509 as they apply to the distribution line message specification:

a) **primitive**;

b) **request**;

c) **indication**;

d) **response**;

e) **confirm**;

f) **service primitive** *(primitive de service)*;

g) **service provider** *(prestataire de service)*;

h) **service user** *(utilisateur d'un service).*

3.3 Autres définitions

Pour les besoins de la présente spécification, les définitions ci-dessous s'appliquent également :

3.3.1 **attribut :** Elément de donnée dont on a défini la signification ainsi que l'ensemble des valeurs qu'il peut prendre.

3.3.2 **donnée :** Toute représentation à laquelle on a donné, ou on peut donner, un sens (par exemple caractères).

3.3.3 **jeu de données :** Objet abstrait représentant un sous-ensemble du VDE, utilisé dans un but spécifique.

3.3.4 **DLPM (Distribution Line Protocol Machine) :** Machine abstraite qui exécute les protocoles définis dans la présente spécification.

3.3.5 **contexte DLMS :** Spécification d'éléments de service associée à une sémantique de communication en usage pendant la durée de vie d'une association d'applications.

3.3.6 **attribut clé :** Attribut d'un type d'objet qui fait partie d'une combinaison d'attributs et qui identifie de façon unique l'objet.

3.3.7 **chargement :** Processus de transfert du contenu d'un jeu de données d'un client de DLMS vers un serveur DLMS.

3.3.8 **objet prédéfini :** Objet qui existe dans VDE et que l'on ne peut pas créer à l'aide des services de DLMS.

3.3.9 **objet normalisé :** Objet dont la définition est produite dans la présente spécification, et dont le champ d'accès est spécifique à VDE ou VAA.

3.3.10 **type :** Description abstraite d'une classe de données, que l'on peut véhiculer par la valeur d'une variable.

3.3.11 **télérapatriement :** Processus de transfert du contenu d'un jeu de données (ou d'une partie de celui-ci) d'un serveur DLMS vers un client DLMS.

3.3.12 **VAA-specific (Virtual Application Association Specific)** - *(spécifique à une VAA)* : Adjectif utilisé pour décrire un objet auquel on ne peut accéder que par une association d'applications qui a, au préalable, créé cet objet VAA.

3.3.13 **variable :** Un ou plusieurs éléments de données, auxquels on se réfère globalement par un seul nom.

3.3.14 **accès aux variables :** Mécanisme d'accès aux variables ou aux éléments de variables définis dans VDE.

3.3.15 **VDE-specific** *(spécifique au VDE)* : Adjectif utilisé pour décrire un objet auquel on peut accéder par toute association d'applications établie dans VDE.

f) **service primitive**;

g) **service provider**;

h) **service user**.

3.3 Other definitions

For the purpose of this specification, the following definitions also apply:

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

3.3.2 **data:** Any representation to which meaning is or might be assigned (for example characters).

3.3.3 **data set:** An abstract object that represents a subset of the capabilities of a VDE which is used for a specific purpose.

3.3.4 **Distribution Line Protocol Machine (DLPM):** An abstract machine that carries out the procedures specified in this specification.

3.3.5 **DLMS context:** A specification of the service elements of DLMS and semantics of communication to be used during the lifetime of an application association.

3.3.6 **Key attribute :** Attribute of an object type which is part of a combination of attributes that uniquely identifies the object.

3.3.7 **loading:** The process of transferring the contents of a data set from a DLMS client to a DLMS server.

3.3.8 **predefined object:** An object that exists at the VDE and that may not be created with DLMS services.

3.3.9 **standardized object:** An instance of an object which scope of access is VDE-specific or VAA-specific and the definition of which is provided in this specification.

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

3.3.11 **uploading:** The process of transferring the contents of a data set (or part of them) from a DLMS server to a DLMS client.

3.3.12 **VAA-specific (Virtual Application Association specific):** An adjective used to describe an object which may be accessed only by the application association which has previously created the VAA object.

3.3.13 **variable:** One or more data elements that are referred all together by a single name.

3.3.14 **variable access:** The access mechanism to variables or variable components defined at the VDE.

3.3.15 **VDE-specific:** An adjective used to describe an object which may be accessed by all the application associations established with the VDE.

3.4 Abréviations

Les abréviations suivantes sont utilisées en DLMS :

(A-1)SAP : Point d'accès au service de la couche inférieure à la couche application (couche A-1)

AA : Association d'applications

ACSE : Eléments de service de contrôle d'association

AE : Entité d'application

AP : Processus d'application

ASE : Eléments de service d'application

ASN.1 : Langage "abstract syntax notation one"

DLMS : Spécification des messages de ligne de distribution

DLPM : Machine de protocole de ligne de distribution

DS : Jeu de données

OSI : Interconnexion des systèmes ouverts

PDU : Unité de données de protocole

SAP : Point d'accès au service

SDU : Unité de données de service

TI : Invocation de tâches

VAA : Association virtuelle d'applications

VDE : Equipement virtuel de distribution

3.5 Conventions

3.5.1 Description des paramètres des services

La présente spécification décrit sous la forme de tableaux les paramètres composant les primitives des services de DLMS. Chaque tableau peut contenir de trois à cinq colonnes décrivant les paramètres des services et les primitives de demande ("Req"), d'indication ("Ind"), de réponse ("Resp") et de confirmation ("Conf"). Les colonnes "Resp" et "Conf" sont absentes quand le service n'est pas un service confirmé.

Dans chaque tableau, un paramètre (ou une de ses parties) figure sur chaque ligne horizontale. Dans la colonne de primitive de service appropriée, on utilise un code pour spécifier le type et l'usage de ce paramètre.

3.4 Abbreviations

The following abbreviations are used in DLMS:

(A-1)SAP : Underlying application service access point with (A-1) representing the layer below the application layer.

AA : Application association

ACSE : Association control service element

AE : Application entity

AP : Application process

ASE : Application service element

ASN.1 : Abstract syntax notation one

DLMS : Distribution line message specification

DLPM : Distribution line protocol machine

DS : Data set

OSI : Open systems interconnection

PDU : Protocol data unit

SAP : Service access point

SDU : Service data unit

TI : Task invocation

VAA : Virtual application association

VDE : Virtual distribution equipment

3.5 Conventions

3.5.1 Service parameter description

This specification uses a tabular format to describe the component parameters of the DLMS service primitives. Each table consists of three to five columns describing the service parameters and the request ("Req"), indication ("Ind"), response ("Resp"), and confirm ("Conf") primitives. The "Resp" and "Conf" columns are absent when the service is not a confirmed service.

In each table, one parameter (or a part of it) is listed on each horizontal line. Under the appropriate service primitive columns, a code is used to specify the type of usage of the parameter.

Les codes utilisés sont les suivants :

M - ce paramètre est obligatoire pour les primitives ;

U - ce paramètre est une option de l'utilisateur, il peut ne pas être fourni, en fonction de l'usage dynamique qui en est fait par l'utilisateur DLMS ;

S - ce paramètre est sélectionné parmi d'autres paramètres (S) en tant que réponse de l'environnement du serveur DLMS ;

C - ce paramètre dépend d'autres paramètres ou de l'environnement de l'utilisateur DLMS ;

(blanc) - ce paramètre n'est jamais utilisé.

Le code "(=)", suivant un des codes M, U, C ou S, indique que ce paramètre est sémantiquement équivalent au paramètre de la primitive de service située immédiatement à sa gauche dans le tableau. (Par exemple, un code "M(=)" dans la colonne de la primitive du service "indication", et un "M" dans celle du service "request" *(demande)*, signifie que ce paramètre dans la primitive "indication" est sémantiquement équivalent à celui de la primitive "request").

Certains paramètres peuvent contenir des sous-paramètres. Les sous-paramètres sont identifiés par les mêmes codes M, U ou C que les paramètres, et ils sont inscrits sous le paramètre. La présence des sous-paramètres dépend toujours de la présence du paramètre sous lequel ils figurent (par exemple, un paramètre optionnel peut avoir des sous-paramètres, si ce paramètre n'est pas fourni; aucun sous-paramètre ne peut l'être).

Les descriptions de paramètres dans cette spécification font référence à des types afin de décrire les valeurs autorisées de tels paramètres. Les types auxquels il est fait référence sont définis dans l'ISO/IEC 8824.

3.5.2 Adressage dans DLMS

La spécification DLMS ne fournit pas de moyens pour nommer et adresser des utilisateurs homologues de DLMS ou des homologues de DLPM. Dans OSI et dans DCP, l'identification et l'adressage des entités homologues d'application sont réalisés à l'aide des services de ACSE définis dans l'ISO 8649. Une fois qu'une telle association d'homologues a été réalisée avec succès, tous les DLMS PDU circulent entre ces homologues en utilisant les services de la couche (A-1). Il n'est donc pas nécessaire que DLMS produise une information d'adressage. On pourra trouver des informations supplémentaires sur la dénomination et l'adressage dans la spécification de la couche application.

The codes used are listed below:

M - the parameter is mandatory for the primitive;

U - the parameter is a user option, and may or may not be provided depending on dynamic usage by the DLMS user;

S - the parameter is selected among other S-parameters as internal response of the DLMS server environment;

C - the parameter is conditional upon other parameters or the environment of the DLMS user;

(blank) - the parameter is never present.

The "(=)" code following one of the M, U, C or S codes indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. (For instance, an "M(=)" code in the indication service primitive column and an "M" in the request service primitive column means that the parameter in the indication primitive is semantically equivalent to the one in the request primitive).

Some parameters may contain subparameters. Subparameters are indicated by labelling of the parameters as M, U or C, and indenting all subparameters under the parameter. Presence of the subparameters is always dependent on the presence of the parameter that they appear under (for example, an optional parameter may have subparameters; if the parameter is not supplied, then no subparameters may be supplied).

The descriptions of parameters in this specification make reference to types, in order to describe the allowable values for such parameters. The types referenced are defined in ISO/IEC 8824.

3.5.2 Addressing in DLMS

The DLMS specification does not provide the means for naming and addressing of a peer DLMS user or peer DLPM. Within OSI and within DCP, the identification and addressing of peer application entities is carried out through the use of ACSE services defined in ISO 8649. After such an association between peers has succeeded, all of the DLMS PDUs flow between these peers over the (A-1)-LAYER services. It is therefore not necessary for DLMS to carry addressing information. Additional information on naming and addressing may be found in the application layer specification.

3.5.3 Conventions de services

La présente spécification applique les conventions de description contenue dans les conventions de service OSI (ISO/IEC/TR 8509). Ces conventions de service OSI définissent les interactions entre les utilisateurs DLMS et les prestataires DLMS. L'information est échangée entre utilisateurs et prestataires DLMS à l'aide des primitives de service qui peuvent véhiculer des paramètres.

Les remarques ci-dessous s'appliquent à l'utilisation de ce modèle :

a) l'ISO/IEC/TR 8509 définit un modèle des services assurés par une couche du modèle de référence OSI. Le service DLMS ne correspond pas à une telle couche (il décrit une partie de la couche application) mais il applique ce modèle dans tous les autres aspects;

b) à tout instant, une AE a de multiples demandes de service en attente, chacune étant traitée indépendamment des autres.

NOTE – Il faut remarquer que la distinction entre un utilisateur DLMS et un prestataire DLMS n'est qu'une abstraction qui peut ne pas nécessairement correspondre à une réalisation DLMS dans un système donné.

3.5.4 Utilisateur DLMS et DLPM en émission et en réception

La présente spécification utilise les termes utilisateur DLMS en émission et en réception. L'utilisateur DLMS en émission est celui qui émet une primitive de service "request" (demande) ou "response". L'utilisateur DLMS en réception est celui qui reçoit une primitive de service "indication" ou "confirm".

NOTE – Il est important de noter que pendant l'exécution d'un service DLMS confirmé, les utilisateurs DLMS seront à la fois récepteurs et émetteurs. Le premier utilisateur émet une demande et reçoit une confirmation, pendant que le second reçoit une indication et émet une réponse.

La présente spécification utilise les termes DLPM en émission et en réception. Le DLPM en émission émet des DLMS PDU. Le DLPM en réception reçoit les DLMS PDU.

3.5.3 Service conventions

This specification uses the descriptive conventions contained in the OSI Service Conventions (ISO/IEC/TR 8509). The OSI service conventions define the interactions between the DLMS user and the DLMS provider. Information is passed between the DLMS user and the DLMS provider by service primitives, which may convey parameters.

The following remarks apply to the use of this model:

a) ISO/IEC/TR 8509 defines a model for the service provided by a layer of the OSI reference model. The DLMS service does not correspond to such a layer (it describes a part of the application layer) but the model used is identical in all other respects;

b) at any instant in time, an AE has multiple service requests outstanding, each proceeding independently of the others.

NOTE – It should be noted that the DLMS user/DLMS provider distinction is an abstraction, and may not necessarily correspond to a realization of DLMS in any particular system.

3.5.4 Sending and receiving DLMS user and DLPM

This specification makes use of the terms sending and receiving DLMS users. The sending DLMS user is the DLMS user that issues a request or response service primitive. The receiving DLMS user is the DLMS user that receives an indication or confirmation service primitive.

NOTE – It is important to note that, in the course of completion of a confirmed DLMS service, both DLMS users will be sender and receiver at the same time. The first DLMS user sends the request and receives the confirmation, while the second DLMS user receives the indication and sends the response.

This specification makes use of the terms sending and receiving DLPMs. The sending DLPM is the DLPM that sends a DLMS PDU. The receiving DLPM is the DLPM that receives a DLMS PDU.

3.5.5 Utilisateur DLMS, demandeur et répondeur

La présente spécification utilise les termes d'utilisateurs DLMS demandeurs et répondeurs. L'utilisateur demandeur est celui qui émet des primitives de service "request" (demande) pour un service, alors que le répondeur est celui qui émet des primitives de service "response" à un service.

NOTE – Il est important de noter que l'usage du terme "utilisateur DLMS répondeur" diffère de l'usage de "entité répondeuse" dans ACSE et d'autres normes. Dans ces normes, ce terme est utilisé pour désigner l'entité qui répond à une demande de connexion.

3.5.6 Clients et serveurs d'un service

La présente spécification utilise les termes de clients et de serveurs afin de décrire un modèle de DLMS VDE. Le serveur est défini comme un système qui se comporte vis à vis de l'extérieur comme un VDE pour une soumission d'une primitive de service "request" (demande) particulière. Le client est un système qui utilise VDE dans un but particulier au travers de soumissions de primitives de service "request". Le VDE sert tout d'abord à décrire les actions du serveur, ensuite à décrire les questions et les réponses dont le client peut user.

3.5.7 Modélisation d'objets

La présente spécification utilise une technique de modélisation abstraite d'objets afin de décrire de manière exhaustive le modèle d'équipement DLMS, ainsi que les procédures DLMS. Cette technique de modélisation décrit les objets abstraits avec leurs caractéristiques et les opérations possibles. Ces objets sont abstraits et aident à la compréhension des intentions et des effets des procédures DLMS. En implémentant DLMS, un système réel applique les concepts décrits dans le modèle sur des unités réelles. De cette façon, vu de l'extérieur, un équipement conforme à cette spécification présente les caractéristiques décrites par la technique de modélisation des objets, mais les mécanismes pour la réalisation de cette apparence ne sont pas fournis dans la spécification.

Le modèle DLMS définit un ensemble d'objets. Chaque objet constitue une entité abstraite qui présente certaines caractéristiques et peut être affectée par certains services de DLMS et par certaines opérations. Chaque type d'objet reçoit un nom par lequel on peut s'y référer.

Chaque type d'objet est caractérisé par un ensemble d'attributs qui servent à décrire des dispositifs visibles de l'extérieur de cet objet. Certains services DLMS peuvent modifier les caractéristiques, lors de l'occurrence d'un objet ; de cette façon, leur influence sur un équipement peut être modélisé par une modification de un ou plusieurs attributs de cet objet.

3.5.5 Requesting and responding DLMS user

This specification makes use of the terms requesting and responding DLMS users. The requesting DLMS user is the DLMS user that issues the request service primitive for a service, while the responding DLMS user is the DLMS user that issues the response service primitive for a service.

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

3.5.6 Client and server of a service

This specification makes use of the terms client and server in order to describe the model of the DLMS VDE. The server is defined as the system that behaves externally as a VDE for a particular service request instance. The client is the system that makes use of the VDE for some particular purpose via a service request instance. The VDE is primarily useful in describing the actions of the server, and thus is describing the requests and responses that a client may make use of.

3.5.7 Object modelling

This specification makes use of a technique of abstract object modelling in order to fully describe the DLMS device model and the DLMS service procedures. In this modelling technique, abstract objects with characteristics and possible operations are described. The objects defined are abstract, and aid in the understanding of the intent of DLMS service procedures and their effects. In implementing DLMS, a real system maps the concepts described in the model to the real device. Hence, as viewed externally, a device that conforms to this specification exhibits the characteristics described in the object modelling technique, but the mechanisms for the realization of this view are not defined by this specification.

The DLMS model defines a set of objects. Each object constitutes an abstract entity which exhibits certain characteristics and may be affected by certain DLMS services and operations. Each object type is given a name, by which it may be referenced.

Each object type is characterized by a set of attributes, which serve to describe some externally visible features of the object. Some DLMS services may modify the characteristics of an object instance, and hence their effect on the device may be modelled by a change in one or more attributes of an object.

Lors de chaque occurrence d'un objet d'un type donné, il doit être identifié de façon unique parmi les autres objets. Pour cela, un des attributs des objets doit être unique (par exemple, beaucoup d'objets ont un attribut "nom" qui est différent pour chaque occurrence d'un objet de ce type). Dans DLMS tout attribut qui rend un objet unique est un attribut clé (Key attribute).

Enfin, certains objets ont des attributs conditionnels en ce sens qu'ils s'appliquent à cet objet si et seulement si certaines conditions sont satisfaites. DLMS exprime de tels attributs en utilisant des "contraintes" qui définissent des conditions. Les attributs soumis à ces contraintes sont considérés comme des attributs de cet objet lors de cette occurrence si et seulement si ces contraintes sont satisfaites.

Dans DLMS, les types d'objet sont définis comme suit :

Object : (name of object)
Key attribute: (name of attribute)
attribute : (name of attribute)
•
•

•
constraint : (constraint expression)
attribute : (name of attribute)
attribute : (name of attribute)
attribute : (name of attribute)

Par convention, toute définition d'objet commence par la déclaration "object" et le nom de cet objet. Ensuite, décalé, le nom d'un attribut clé. Puis sont cités zéro, un ou plusieurs attributs. Les contraintes peuvent être exprimées n'importe où entre les attributs. Il est convenu que les attributs soumis à cette contrainte sont décalés. Le premier attribut qui n'est pas décalé ferme la liste des attributs soumis à la contrainte.

Each object instance within an object type must be uniquely identified among all objects. For this purpose, one of the objects attributes must be unique (for example, many objects have a name attribute, which is different for each object instance of the object type). In DLMS, each attribute which makes the object unique is identified as a "Key attribute".

Finally, some objects contain attributes which are conditional, in the sense that they are relevant to the object if and only if certain conditions are true. DLMS expresses such attributes through the use of "constraint", which specify a condition. Attributes that are subject to a constraint are considered as object attributes for an object instance if, and only if, the constraint is satisfied for that object instance.

In DLMS, object types are syntactically defined as follows:

Object: (name of object)
    Key attribute: (name of attribute)
    attribute: (name of attribute)
    •
    •

    •
    constraint: (constraint expression)
        attribute: (name of attribute)
        attribute: (name of attribute)
    attribute: (name of attribute)

By convention, each object definition begins with an object declaration and the name of the object. Immediately following and indented, one Key attribute is named. Next, zero, one or more attributes are named. Constraints may be expressed anywhere within the attributes, with the convention that all the constrained attributes are indented underneath it. The first attribute definition that is not indented ends the list of attributes that are subject to the constraint.

4 Equipement virtuel de distribution

4.1 Relation entre VDE et le modèle OSI

4.1.1 VDE dans les AP

VDE existe dans DLMS AP. Il constitue la partie de la tâche de traitement de l'information qui rend disponible, pour la transmission d'une information intelligible, un ensemble de ressources et de fonctionnalités associées à un équipement réel de distribution.

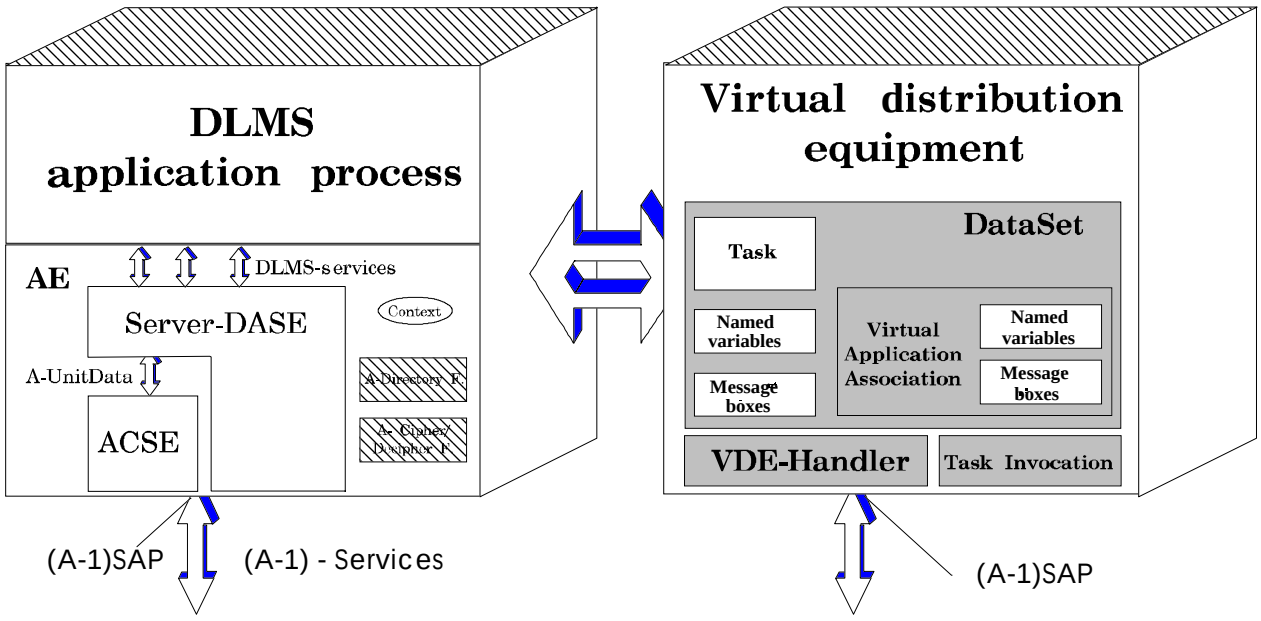

Figure 1 - Modèle d'équipement de distribution

Un AP peut définir zéro, un, ou plusieurs équipements virtuels de distribution. S'il ne définit pas au moins un VDE, il ne peut pas jouer le rôle de serveur DLMS.

Chaque VDE représente un équipement virtuel de distribution dans un AP; cela modélise une partie cohérente du comportement de l'AP. Chaque VDE est séparé logiquement des autres VDE.

4 Virtual distribution equipment

4.1 Relationship of the VDE to the OSI model

4.1.1 The VDE within the AP

A VDE exists within the DLMS AP. It constitutes that portion of an information processing task which makes available, for transmission of meaningful information, a set of resources and functionalities associated with a real distribution device.

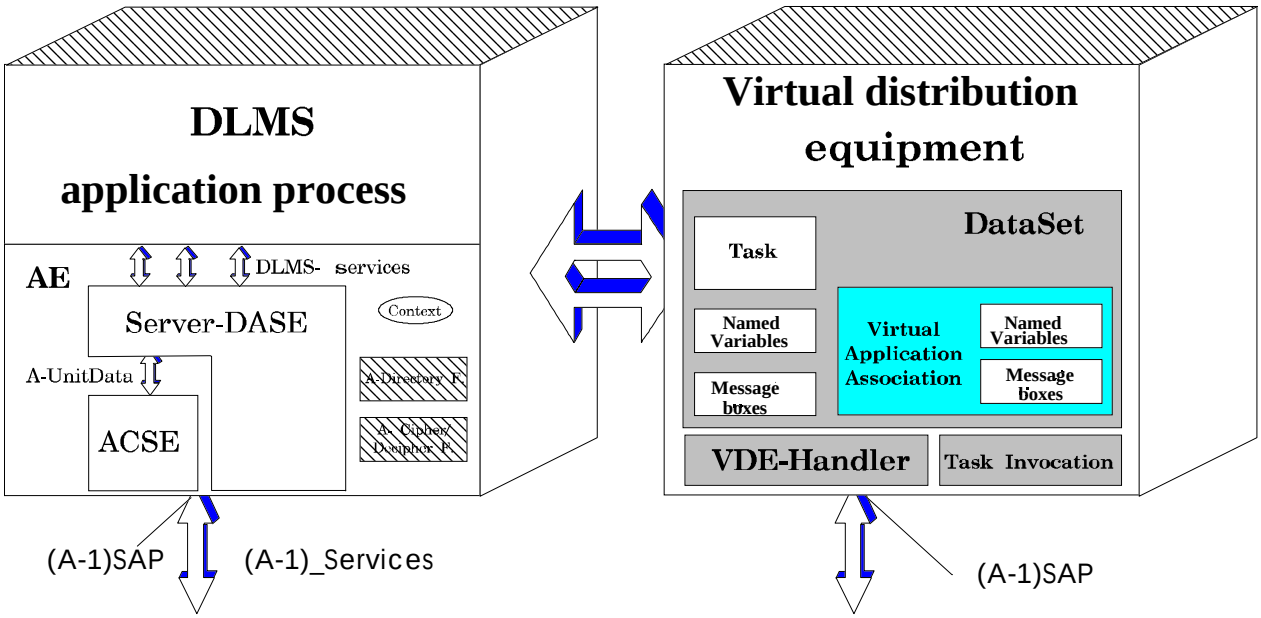

Figure 1 - Distribution equipment model

An AP may define zero, one or more virtual distribution equipments. If it does not define at least one VDE it may not act as a DLMS server.

Each VDE represents a virtual distribution device within an AP that models a coherent part of the AP behaviour. Each VDE is logically separated from all the other VDEs.

Exemple :

Un système DLMS connecté à un environnement non-DLMS composé de multiples unités de distribution telles que compteurs, interrupteurs, unités de contrôle, peut être modélisé :

- comme un AP unique contenant un VDE pour chacun des équipements;

- comme plusieurs AP contenant chacun un seul VDE distinct pour chaque équipement.

Dans tous les cas, les clients des VDE verront chacun des équipements comme un VDE unique. Par rapport aux services de DLMS, ce VDE apparaîtra comme étant indépendant de tous les autres VDE.

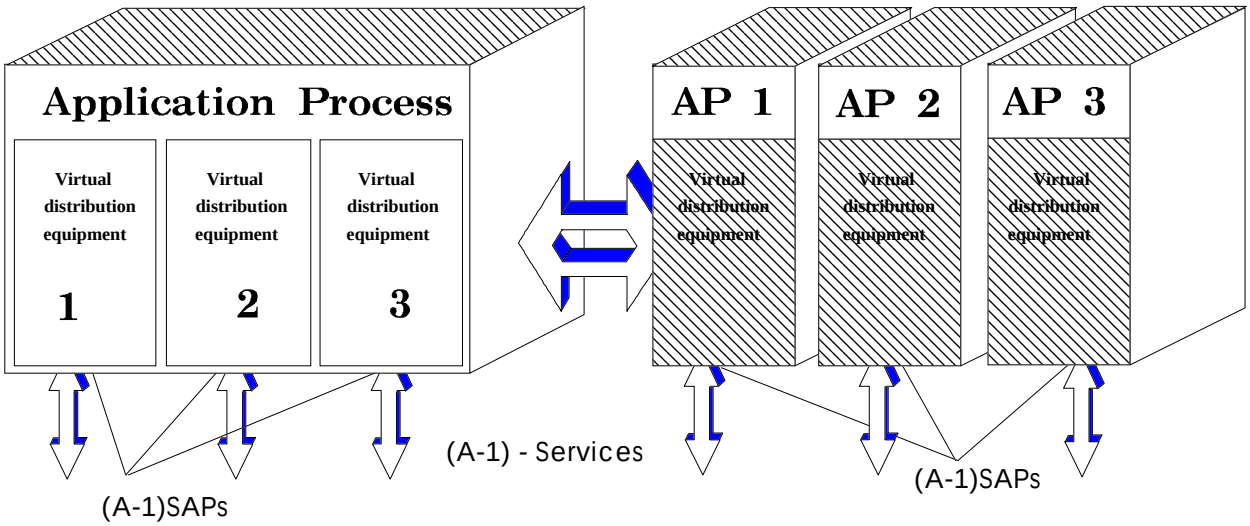

Figure 2 - VDE dans les AP

Etant donné qu'un ensemble de plusieurs AP contenant chacun un seul VDE est équivalent à un seul AP contenant plusieurs VDE, on supposera pour la suite qu'un AP ne contient qu'un seul VDE.

Example:

A DLMS system which is connected to a non-DLMS environment containing multiple attached distribution devices like meters, switches or control units could be modelled:

- as a single AP containing one VDE for each attached device;

- as several application processes, each containing a single, distinct VDE for each attached device.

The clients of the VDEs in each case will see a particular attached device as a single VDE. Relative to the DLMS services, this VDE appears to be independent of all the other VDEs.

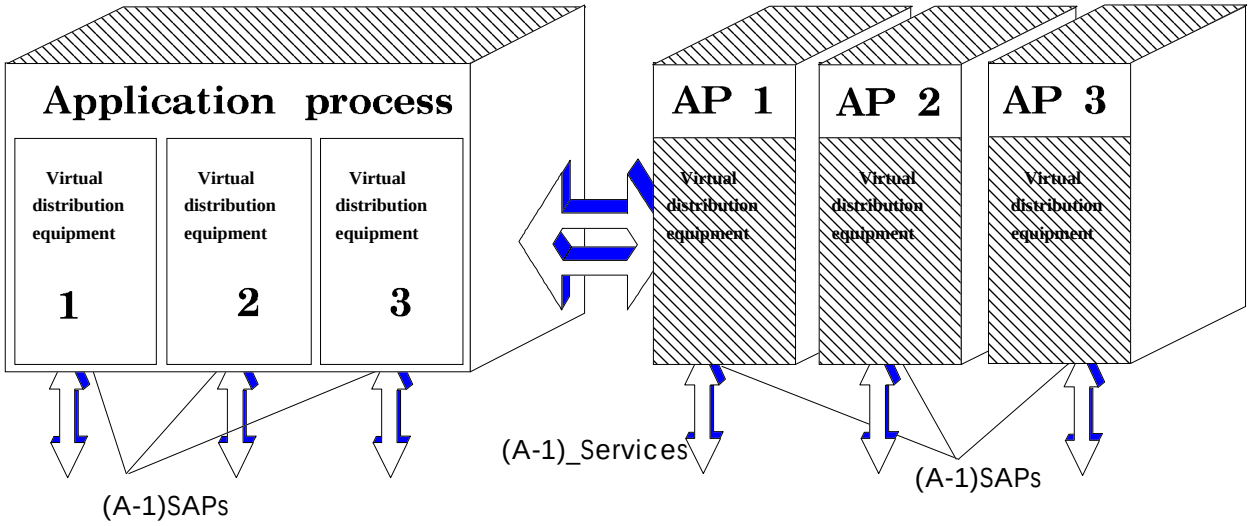

Figure 2 - VDE within APs

According to the equivalence between several APs containing each a single VDE and a single AP containing several VDEs, it is assumed in the following that an AP contains only one VDE.

4.1.2 L'AE dans le VDE

Comme le décrit l'ISO 7498, une AE représente un ensemble de capacités de communication de l'AP. Une AE modélise les aspects d'un AP qui doivent être considérés pour les besoins de la communication OSI. Un AP sans AE ne peut pas communiquer dans un environnement OSI ni dans un environnement DCP. Pour plus de détails, se reporter à la spécification de la couche application de DCP ou à l'ISO 7498, l'ISO/CEI 9545 et l'ISO/CEI 8649.

Chaque VDE peut contenir une ou plusieurs AE. Chaque AE représente les capacités de communication de la partie de l'AP modélisée dans le VDE. Un VDE qui ne contiendrait pas de AE ne peut pas communiquer dans l'environnement DCP. Chaque AE fait partie d'un et seulement un VDE. La communication avec une AE donnée est utilisée pour modéliser la communication avec un VDE, et par là, avec un AP.

NOTES

1 DLMS suppose qu'il n'existe, dans un AP, qu'une seule AE d'un type donné correspondant à une (A-1)SAP, et donc aussi dans un VDE. Si une autre AE d'un autre type existe dans l'AP, elle doit utiliser une autre adresse (A-1)SAP. Cette configuration ne sera pas considérée par la suite.

2 A partir de la couche sous-jacente, chaque AE est identifiée par une seule adresse (A-1)SAP et peut donc être adressée conformément à l'OSI.

Une AA est une relation coopérative entre deux AE. Elle fournit le cadre de référence nécessaire à un interfonctionnement effectif entre les AE.

Une invocation de AE est une utilisation spécifique des capacités de l'AE dans un cas de communication. Chaque invocation d'AE modélise un cas d'usage de cette AE dans une AA. On utilise plusieurs invocations d'AE pour représenter plusieurs associations d'applications.

NOTE 3 – DLMS ne permet pas plusieurs associations d'applications dans une seule invocation d'AE. Des AA multiples sont modélisées par des invocations d'AE multiples.

Un VDE donné est adressé par un seul (A-1)SAP. Cette association d'un (A-1)SAP et d'un VDE est de longue durée. L'équivalence entre un AP agissant en tant que serveur DLMS et son VDE est illustrée à la figure 3.

4.1.2 The AE within the VDE

As described in ISO 7498, an AE represents a set of communication capabilities of the AP. An AE models the aspects of an AP which need to be taken into account for the purpose of OSI communications. An AP without AE cannot communicate in the OSI environment or in the DCP environment. Please refer to the DCP application layer specification or to the ISO 7498, ISO/IEC 9545 and ISO/IEC 8649 standards for further details.

Each VDE may contain one or more AEs. Each AE represents the set of communication capabilities of the AP part that is modelled with the VDE. A VDE that contains no AE cannot communicate in the DCP environment. Each AE is a part of one and only one VDE. Communication with a particular AE is used to model communication with a VDE, and hence with an AP.

NOTES

1 DLMS assumes that there is only one AE of a specified type, corresponding to a particular (A-1)SAP in an AP and thus in a VDE. If an AE of another AE-type exists within the AP, it shall use another (A-1)SAP address. Such a configuration is not further taken into account.

2 From the underlying layer, each AE is identified by a single (A-1)SAP address and may be therefore consistently addressed within OSI.

An AA is a cooperative relationship between two AEs. It provides the necessary frame of reference between the AEs in order that they may interwork effectively.

The AE-invocation represents a specific use of the capabilities of an AE in an instance of communication. Each AE-invocation models an instance of usage of that AE in an AA. Several AE-invocations of an AE are used to represent several application associations.

NOTE 3 – DLMS does not allow multiple application associations in a single AE-invocation. Multiple AAs are modelled as multiple AE-invocations.

A given VDE is addressed through an AE by a single (A-1)SAP. This binding of an (A-1)SAP to a VDE is of long duration. The equivalence between an AP acting as a DLMS server and its VDE is shown in figure 3.

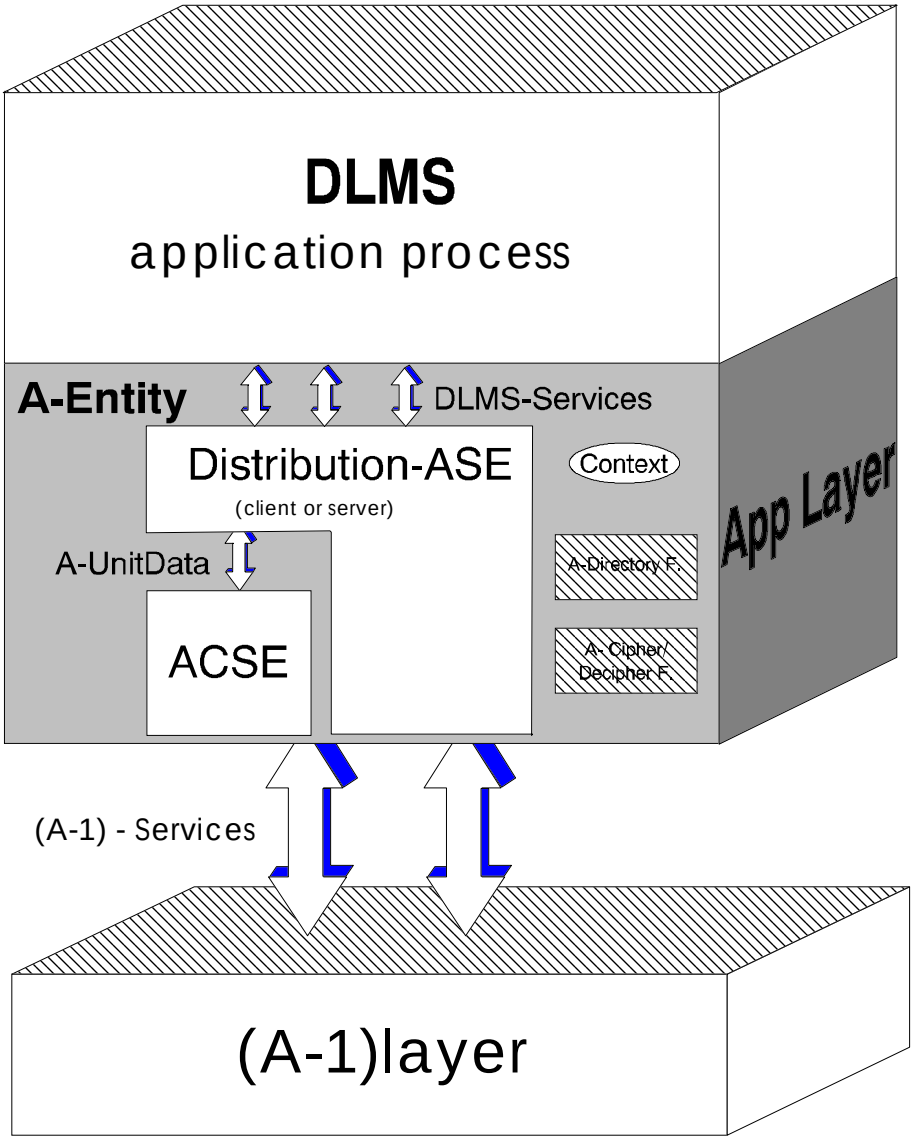

Figure 3 - Structure de processus d'application

4.2 Relation avec un équipement réel de distribution

Un VDE est une représentation abstraite d'un ensemble spécifique de ressources et de fonctionnalités d'un équipement réel ainsi que l'application de cette représentation abstraite aux aspects fonctionnels physiques d'un équipement de distribution réel. Cette application d'une ressource virtuelle à la ressource réelle sous-jacente est de longue durée (comparable en fait à la durée de vie du matériel).

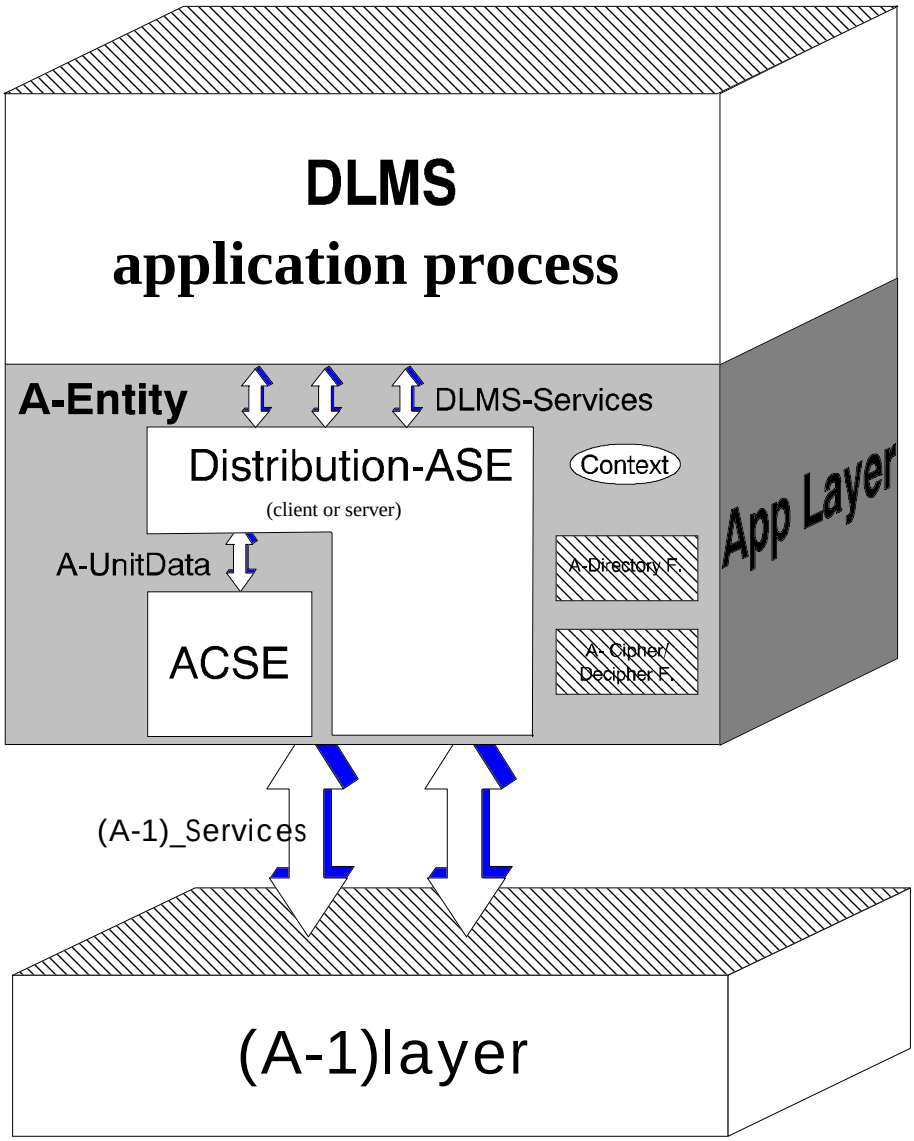

Figure 3 - Application process structure

4.2 Relationship with a real distribution device

A VDE is an abstract representation of a specific set of resources and functionalities of a real distribution device and a mapping of this abstract representation to the physical and functional aspects of the real distribution device. This mapping of a virtual resource to the underlying real resource is of long duration (comparable in fact to the lifetime of the hardware).

En général, les ressources d'un VDE donné sont différentes et indépendantes des ressources des autres VDE. Quand les ressources virtuelles de plusieurs VDE sont appliquées sur la même ressource physique sous-jacente, il faut que les processus d'application fournissent un mécanisme de coordination pour coordonner leur accès à cette ressource unique. Un sémaphore contrôle habituellement l'accès à une ressource réelle partagée. Les sémaphores ne sont pas décrits dans DLMS. La manipulation des sémaphores est considérée comme un problème local.

NOTE – Un objet "variable désignée" peut être utilisé pour représenter l'état d'un sémaphore dans VDE.

DLMS décrit les opérations d'un VDE en décrivant les objets abstraits qu'il manipule. Il décrit aussi l'ensemble des opérations qui peuvent être réalisées sur ces objets à l'aide des services de DLMS.

4.3 Structure d'un VDE

4.3.1 Vue d'ensemble de la structure

Chaque VDE contient exactement :

- un VDE-handler (gestionnaire VDE) ;
- un jeu de données ;
- zéro, une ou plusieurs TI ;
- zéro, une ou plusieurs variables.

La figure 4 ci-dessous décrit les parties constituant un VDE.

Generally, the resources of a given VDE are distinct from, and independent of, the resources of all the other VDEs. When virtual resources of several VDEs are mapped to the same underlying physical resource, a mechanism shall be provided by the application processes to co-ordinate their accesses to the single resource. A semaphore usually controls access to the real shared resource. Semaphores are not further discussed in DLMS. The manipulation of real semaphores is considered to be a local issue.

NOTE – A "named variable" object can be used to represent the states of a semaphore within a VDE.

DLMS describes the operation of a VDE by describing the abstract objects which are manipulated by it. It also describes the set of operations which may be performed on these objects through the use of DLMS services.

4.3 Structure of a VDE

4.3.1 Structure overview

Each VDE contains precisely:

- one VDE-handler;
- one data set;
- zero, one or more TIs;
- zero, one or more variables;

The following figure 4 shows the constituent parts of a VDE.

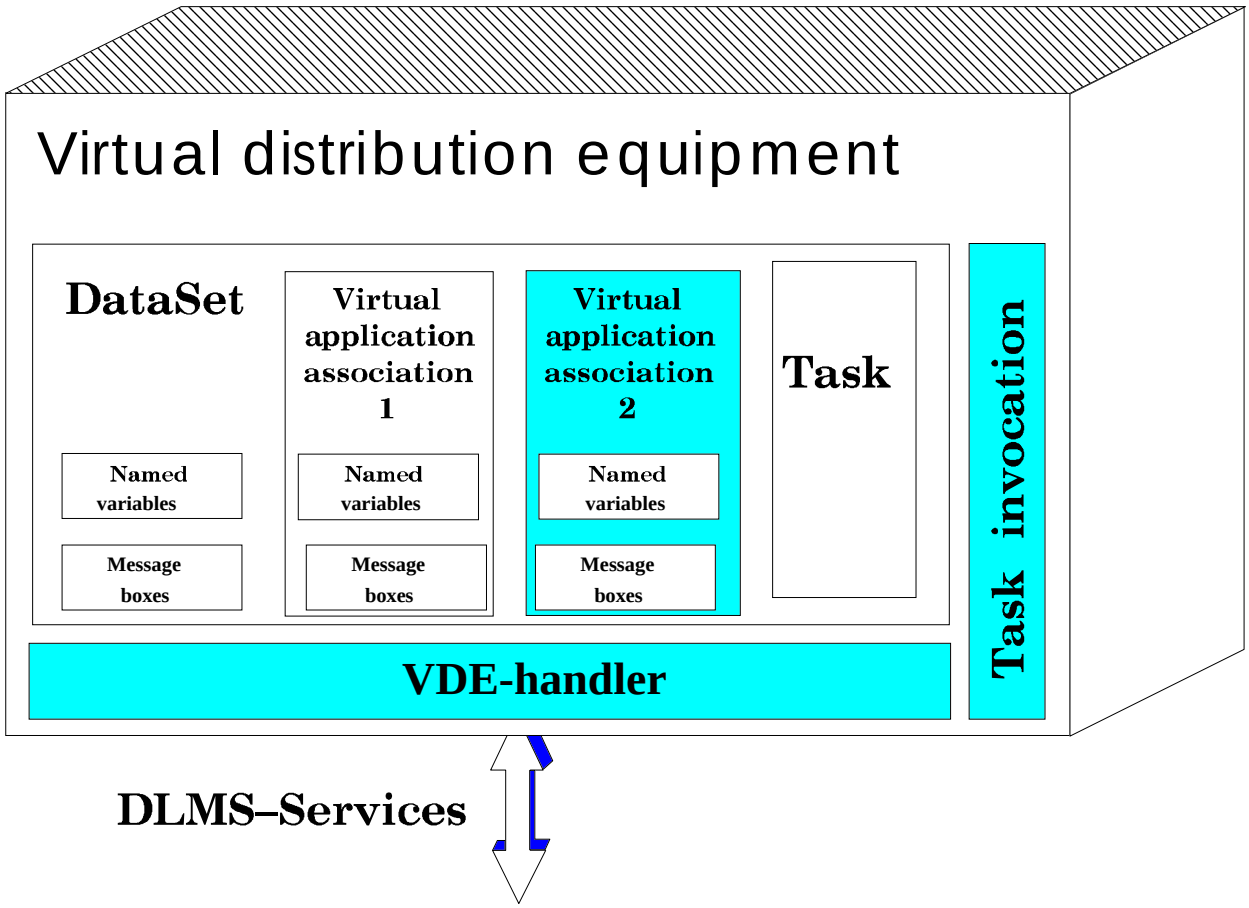

Figure 4 - Structure du VDE

4.3.2 Le VDE-handler (gestionnaire VDE)

Le VDE-handler gère tous les accès utilisant les services de DLMS, aux fonctions intégrées dans un VDE. Il représente l'interface entre l'environnement DCP et l'environnement local d'exploitation du VDE. Conceptuellement, le VDE-handler inclut tous aspects visibles par les clients, de l'exploitation locale du VDE, sauf ceux qui sont explicitement définis dans le jeu de données. Ainsi, le VDE-handler supporte la visibilité de l'environnement local d'exploitation ainsi que des procédures définies localement et qui peuvent être invoquées à l'aide de DLMS.

Le VDE-handler contient les ressources du VDE. Certaines de ces ressources sont utilisées par le jeu de données.

Le VDE-handler représente les fonctions incluses dans le VDE nécessaires pour se comporter comme un VDE. L'existence d'un VDE-handler pleinement opérationnel se confond exactement avec l'existence d'un VDE.

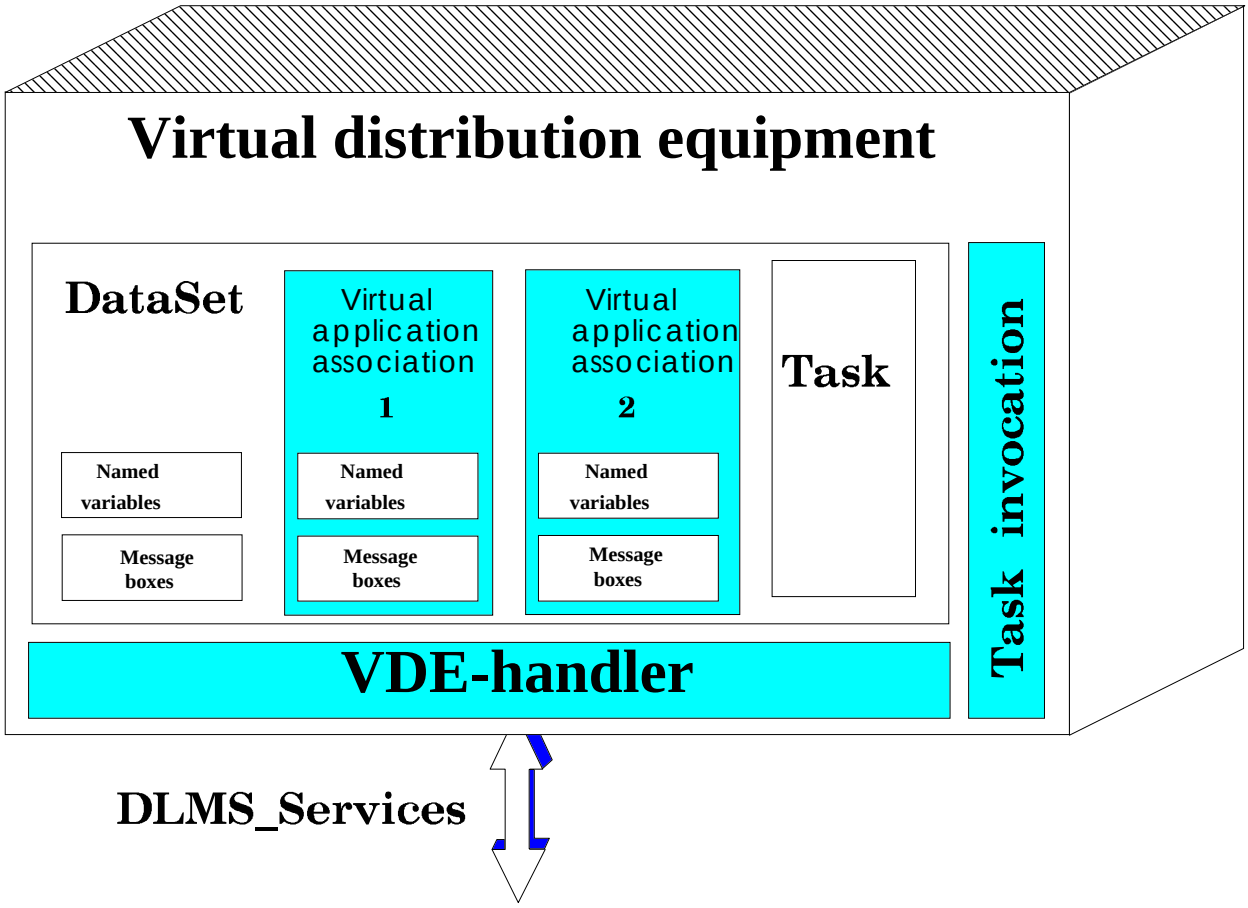

Figure 4 - The VDE structure

4.3.2 The VDE-handler

The VDE-handler manages all access using the DLMS services to the built-in resources of a VDE. It represents the interface between the DCP environment and the local operating environment of the VDE. Conceptually, the VDE-handler includes all aspects of the local operation of the VDE which are visible to its clients, except for those aspects which are explicitly defined by the data set. Thus, the VDE-handler supports DLMS visibility to the local operating environment as well as visibility to locally defined procedures which may be invoked using DLMS.

The VDE-handler contains the resources of the VDE. Some of the resources are used by the data set.

The VDE-handler represents the built-in functions of the VDE which are necessary for it to act as a VDE. The existence of a fully functional VDE-handler corresponds exactly with the existence of the VDE.

4.3.3 Le jeu de données

Dans DLMS, l'objet jeu de données représente l'ensemble des informations liées à la destination spécifique du VDE. Le jeu de données peut être vide ou peut contenir des informations. Les informations peuvent être des instructions de programmes, des tables de valeurs, différentes listes de variables désignées ou des messages spéciaux. Le jeu de données, en tant qu'ensemble d'informations, peut être chargé à l'aide des services de DLMS.

Plus conceptuellement, le jeu de données représente les ressources du VDE utilisées pour la réalisation de la destination spécifique du VDE. Le jeu de données inclut les aspects du VDE associés à une stratégie de distribution coordonnée.

L'allocation des ressources du VDE à son jeu de données est statique. Les ressources du jeu de données sont prédéfinies dans le serveur DLMS et ne peuvent pas être modifiées à l'aide des services de DLMS.

Les objets variables peuvent être définis et subordonnés au jeu de données, qui représente alors un espace unique de noms pour ces variables DLMS.

Un AP peut charger le contenu du jeu de données à l'aide des services de gestion des jeux de données.

4.3.4 L'invocation de tâches

La TI est un élément qui correspond très exactement à la mise d'un programme dans la queue d'exécution dans un environnement multitâches. Une TI est associée à une tâche qui contient le code à exécuter avec les variables associées. Une TI contient toutes les informations nécessaires pour initialiser l'exécution de la tâche correspondante dans le VDE.

Dans DLMS, la TI est un des objets prédéfinis du VDE. La tâche correspondante peut être chargée à partir du contenu d'un jeu de données. On pourra trouver plus de détails sur les tâches dans les autres sections de la CEI 1334.

Un AP peut manipuler les TI en utilisant les services de gestion des invocations de tâches.

4.3.5 Les variables

La variable est un objet abstrait de VDE capable de fournir (quand on le lit) ou d'accepter (quand on écrit) une valeur d'une donnée typée. Un type de données (ou simplement type) est une description abstraite de la classe de données qui peut être véhiculée par la valeur d'une variable. Un type de variable détermine sa syntaxe abstraite, la gamme des valeurs possibles et sa représentation lors de la communication à l'aide de DLMS.

4.3.3 The data set

In DLMS, the data set object represents the set of information that is bound to a specific purpose of the VDE. The data set may be empty or may contain information. The information may be program instructions, values tables, various lists of named variables or special messages. The data set, as a set of information, may be loaded using DLMS services.

More conceptually, the data set represents the set of VDE resources which is used for achieving a specific purpose of the VDE. The data set includes those aspects of the VDE which are associated with a co-ordinated distribution strategy.

The allocation of the VDE resources to its data set is static. The data set resources are predefined within the DLMS server and cannot be changed using DLMS services.

Variable objects may be defined subordinated to the data set, which then represents a single name space for those DLMS variables.

An AP can load the contents of the data set by using the data set management services.

4.3.4 The task invocation

The TI is an element which most closely corresponds to putting a program on the execution queue in a multi-tasking environment. A TI is associated with a task that contains the code to be executed and the associated variables. A TI contains all the information necessary to initiate the execution of the corresponding task within the VDE.

In DLMS, the TI is one of the predefined objects at the VDE. The corresponding task itself may be loaded with a data set content. Further details on tasks may be found in other sections of IEC 1334.

An AP can manipulate the TIs by using the task invocation management services.

4.3.5 The variable

The variable is an abstract element of the VDE which is capable of providing (when read) or accepting (when written) a typed data value. A data type or simply type is an abstract description of the class of data which may be conveyed by a variable value. A variable type determines its abstract syntax, its range of possible values, and its representation while being communicated using DLMS.

4.4 Spécification des objets DLMS

Le présent paragraphe définit certains attributs fondamentaux des objets DLMS.

4.4.1 Noms des objets

Les objets DLMS sont référés par un nom. Dans DLMS, l'accès à un objet n'est possible que par l'usage de son nom. Il n'existe pas actuellement dans DLMS de moyen d'accès direct à une ressource réelle. Les noms des objets sont destinés à être les attributs clés des objets DLMS. Le nom d'un objet doit donc être unique dans son espace de définition. Cet espace est appelé champ de désignation de l'objet.

4.4.2 Champ de désignation

Dans DLMS, on définit seulement un champ de désignation : la largeur de VDE. Le champ de désignation est donc le même pour tous les objets d'un VDE. Un nom d'objet doit être défini de façon unique dans le VDE sans tenir compte de la classe d'objet. N'importe quel client du VDE peut se référer au nom de l'objet.

4.4.3 Champ d'accès

Le champ d'accès définit les restrictions à l'accessibilité à un objet d'un VDE, cela afin d'éviter une manipulation non autorisée de l'objet et la confidentialité des données dans un VDE.

Dans DLMS, deux champs d'accès sont définis, les champs "VDE-specific" (spécifiques à VDE) et "VAA-specific" (spécifiques à VAA).

Un objet dans un champ d'accès VDE-spécific peut être manipulé par tous les utilisateurs DLMS homologues du VDE. Par exemple, une variable désignée dans le champ d'accès VDE-specific est accessible librement. Chaque utilisateur DLMS homologue peut vouloir lire la variable.

Un champ d'accès VAA-spécific est lié directement à l'existence de l'objet VAA correspondant. Le champ VAA-spécific définit un champ d'accès virtuel, ouvert à l'utilisateur DLMS qui a, au préalable, créé l'objet VAA, et fermé à tous les autres utilisateurs DLMS.

4.4 Specification of DLMS objects

This subclause defines some basic attributes of the DLMS objects.

4.4.1 Object name

DLMS objects are referenced by name. In DLMS, access to an object is only possible through the use of its name. Actually, there is no means to make a direct raw access to a real resource within DLMS. Object names are intended to be the Key attributes of the DLMS objects. The name of an object shall then be unique within its space of definition. This space is known as the scope of name of the object.

4.4.2 Scope of name

In DLMS, only one scope of name is defined: VDE width. The scope of name is then the same for all the objects in a VDE. An object name shall be uniquely defined within the VDE regardless to its object class. An object name may be referenced by every client of the VDE.

4.4.3 Scope of access

The scope of access defines the restriction of accessibility to an object of a VDE instance. This is to avoid non-authorized manipulation and confidentiality of data in a VDE.

In DLMS, two scopes of access are defined: VDE-specific and VAA-specific.

An object with a VDE-specific scope of access may be manipulated by all the peer DLMS users of that VDE. For example a named variable with a VDE-specific scope is freely accessible. Every peer DLMS user may attempt to read the variable.

A VAA-specific scope of access is directly bound to the existence of a corresponding VAA object. The VAA-specific scope defines a virtual space of access that is open to the DLMS user which has previously created the VAA object and is closed to all the other DLMS users.

Le champ d'accès des objets DLMS peut être modifié à l'aide du service "ChangeScope" (modifier champ) décrit dans l'article 8.

4.4.4 Durée de vie

La durée de vie d'un objet indique si cet objet est défini et subordonné au VDE ou au jeu de données. Dans DLMS, le VDE a une structure statique; un objet défini et subordonné au VDE existe aussi longtemps que le VDE existe. Sa durée de vie est alors la même que celle du VDE. Le jeu de données peut être chargé à l'aide des services de DLMS. La nature du jeu de données est donc plus dynamique. Un objet défini subordonné au jeu de données existe aussi longtemps que le jeu de données existe. Sa durée de vie est alors celle du jeu de données. Dans un VDE, un objet subordonné au jeu de données peut apparaître ou disparaître quand on recharge le jeu de données.

La durée de vie des objets de DLMS est précisée dans l'attribut lifetime (durée de vie).

4.4.5 Classes d'objets

Les occurrences spécifiques des objets DLMS qui peuvent être nommés sont énumérés dans le tableau ci-dessous. Les combinaisons autorisées sont indiquées, associées à la durée de vie afférente aux objets.

Tableau 1 - Classes d'objets

Classes d'objets	VDE-specific	VAA-specific	Durée de vie
Named Variable objects	x	x	DS/VDE
Named Variable List objects	x	x	DS/VDE
Message Box objects	x	x	DS/VDE
TI objects	x	x*	VDE
VAA objects	x		VDE
Data Set object	x	x*	VDE

* Si le champ d'accès d'un jeu de données est VAA-specific, alors tous les objets TI ont le même champ VAA-specific. Si le champ d'accès d'un objet TI est VAA-specific, alors tous les autres objets TI et le jeu de données ont le même champ d'accès VAA-specific. L'objet VAA correspondant est connu comme Executive-VAA.

The scope of access of a DLMS object may be changed through the use of the ChangeScope service described in clause 8.

4.4.4 Lifetime

The lifetime of an object expresses whether an object is defined subordinated to the VDE or to the data set. In DLMS, the structure of the VDE is static; an object defined subordinated to the VDE exists as long as the VDE exists. Its lifetime is then the same as the VDE. The data set may be loaded using DLMS services. The nature of the data set is then more dynamic. An object defined subordinated to the data set exists as long as the data set exists. Its lifetime is then the same as the data set one. At a VDE, an object subordinated to the data set may then appear or disappear when the data set is loaded again.

The lifetime of a DLMS object is specified in the lifetime attribute.

4.4.5 Object classes

The specific instances of DLMS objects which can be named are listed in the following table. The allowed combinations are indicated in association with the inferred lifetime of the objects.

Table 1 - Object classes

Object class	VDE-specific	VAA-specific	Lifetime
Named Variable objects	x	x	DS/VDE
Named Variable List objects	x	x	DS/VDE
Message Box objects	x	x	DS/VDE
TI objects	x	x*	VDE
VAA objects	x		VDE
Data Set object	x	x*	VDE

* If the scope of access of the data set is VAA-specific, then all the TI objects have the same VAA-specific scope. If the scope of access of a TI object is VAA-specific, then all the other TI objects and the data set have the same VAA-specific scope. The associated VAA object is known to be the Executive VAA.

Tous les objets inscrits ci-dessus ont, dans le VDE, une durée de vie que l'on peut déduire de leur type. La durée de vie des objets VDE est aussi longue que celle du VDE, sauf si elle est expressément modifiée. Par exemple, les objets du jeu de données ont la durée de vie du VDE car ils existent dans le VDE tant que le VDE existe, sauf si on les modifie explicitement à l'aide des services de gestion du jeu de données. La durée de vie des objets contenus dans le jeu de données est aussi longue que celle du jeu de données.

4.4.6 Le paramètre "Object Name" (nom d'objet)

Le paramètre Object Name (nom d'objet) est souvent utilisé dans la spécification des services de DLMS. Le paramètre Object Name identifie de manière unique un objet DLMS dans un VDE. Object Name comprend dans sa structure la classe de l'objet. Cela rend plus facile les opérations du VDE-handler (gestionnaire VDE).

La structure du paramètre Object Name est décrite ci-dessous :

Object name
Attribute : object class (NAMED-VARIABLE, NAMED-VARIABLE-LIST,
MESSAGE-BOX, TI , DATA-SET, VAA)
Attribute: Item Identifier

Le paramètre Object Class (classe d'objets) identifie la classe à laquelle l'objet appartient. C'est l'une des classes définies dans DLMS. Le paramètre Item Identifier (identificateur d'item) n'identifie que le nom de l'objet dans une classe particulière d'objets.

L'object name est toujours considéré comme un tout. La construction sémantique des objets n'est fournie que pour garantir la compréhension de la manipulation des objets DLMS. Il doit être clair que la référence à un objet à l'aide de son nom fournit aussi l'identification de sa classe. La description en ASN.1 du paramètre Object Name est fournie dans l'annexe A comme un "integer 16" (entier à 16 bits).

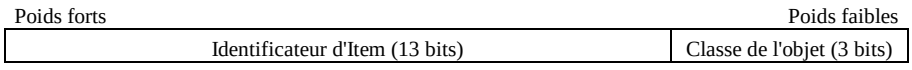

Figure 5 - Structure du paramètre nom d'objet

Les valeurs des object class sont les suivantes :
000 Named Variable object (objet variable désignée) ;
001 Named Variable List object (objet liste de variables désignées);
010 Message Box object (objet boîte à messages) ;
011 TI object (objet TI) ;
100 Data Set object (objet jeu de données) ;
101 réservé ;
110 réservé ;
111 VAA object (objet VAA).

Les valeurs de l'Item Identifier (idendificateur d'item) sont exprimées sur 13 bits. Ces valeurs sont attribuées librement.

Each of the objects listed above has a lifetime within the VDE which can be inferred from its type. The VDE lifetime objects exist as long as the VDE exists unless explicitly changed. For example, the data set object has a VDE lifetime because it exists in the VDE as long as the VDE exists unless explicitly changed with the data set management services. The data set lifetime objects are contained within the data set and exist as long as the data set exists.

4.4.6 Object Name parameter

The parameter Object Name occurs frequently in the specification of DLMS services. The Object Name parameter identifies uniquely within a VDE a DLMS object. The Object Name contains in its structure the class of the object. This makes VDE-handler operations easier.

The structure of the Object Name parameter is described as follow:

Object name
attribute: object class (NAMED-VARIABLE, NAMED-VARIABLE-LIST, MESSAGE-BOX, TI, DATA-SET, VAA)
attribute: Item Identifier

The Object Class parameter identifies the class to which the object belongs. It is one of the DLMS defined object class. The Item Identifier parameter, identifies uniquely the object name within the VDE for a particular object class.

The Object Name is always referenced as a whole. The semantic construction of the object name is provided only to ensure comprehensive manipulation of the DLMS objects. It shall be clear that reference to an object using its name provides also identification of its class. The ASN.1 description of the Object Name parameter is given in annex A as an Integer16 .

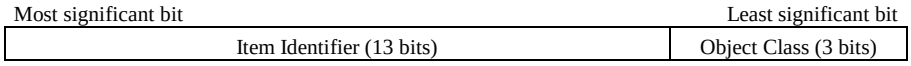

Figure 5 - Object name structure

Object class values are chosen as follows :

000 Named Variable object;
001 Named Variable List object;
010 Message Box object;
011 TI object;
100 Data Set object;
101 reserved;
110 reserved;
111 VAA object.

The Item identifier values are described on 13 bits. Values may be freely chosen.

4.4.7 La description des objets DLMS

Pour la plupart des objets DLMS, la description commence comme suit :

Class of object
key attribute : Object Name
attribute : Scope Of Access (VDE-specific, VAA-specific)
contrainte Scope Of Access = VAA-specific
attribute : VAA name
attribute : Scope May Change (TRUE, FALSE)
attribute : Lifetime (VDE, DATA-SET)
attribute : ...

L'attribut clé est toujours le nom de l'objet. Il n'est fourni que pour identifier l'objet DLMS parmi les autres objets définis dans le VDE, sans se préoccuper de la classe.

Le paramètre Scope of Access (champ d'accès) précise le champ d'accès actuel pour l'objet considéré. Ce champ d'accès définit les règles d'accès à l'objet. Il doit être VDE-specific ou VAA-specific. Si la valeur du champ d'accès est VAA-specific, le nom du VAA est spécifié dans l'attribut Name (nom) du VAA.

L'attribut ScopeMayChange (le champ peut changer) indique s'il est possible ou non de changer le champ d'accès de l'objet considéré à l'aide du service DLMS ChangeScope (changement de champ).

L'attribut Lifetime (durée de vie) spécifie le type de durée de vie de l'objet décrit. Elle doit être VDE ou jeu de données. Un attribut Lifetime dont la valeur est Data Set (jeu de données) signifie que cet objet est défini dans le jeu de données.

Les autres attributs sont plus spécifiques à la classe d'objets.

Altération d'objets DLMS: tous les objets DLMS possèdent, dans leur description, une liste d'attributs visibles de l'extérieur. En plus des services DLMS qui altèrent ces attributs, ceux-ci peuvent être également changés par l'intermédiaire d'une action locale du système ou des opérateurs du système, ou par l'exécution d'une invocation de tâche (task invocation). Les moyens d'altération locale des attributs d'un objet ne sont pas inclus dans cette norme.

4.4.7 DLMS object description

Most of the DLMS object descriptions begin as follows:

Class of object
- key attribute: Object Name
- attribute: Scope Of Access (VDE-specific, VAA-specific)
- constraint Scope Of Access = VAA-specific
  - attribute: VAA Name
- attribute: Scope May Change (TRUE, FALSE)
- attribute: Lifetime (VDE, DATA-SET)
- attribute: ...

The key attribute is always the object name. It is provided to uniquely identify the DLMS object among all other objects defined at the VDE and regardless of its class.

The Scope Of Access parameter defines the current scope of access for the designated object. This scope of access rules the object access. It shall be VDE-specific or VAA-specific. If the Scope Of Access attribute contains the VAA-specific value, the name of the VAA is specified in the VAA Name attribute.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The lifetime attribute specifies which kind of lifetime the described object has. It shall be VDE or DATA-SET. A lifetime attribute set to DATA-SET indicates that the object is defined within the data set.

Other attributes are more specific to the object class.

Alteration of DLMS objects : All DLMS objects have as part of their description a list of externally visible attributes. In addition to the DLMS services which alter these attributes, they may also be changed through the local action of the system or of the system operators, or through the execution of a task invocation. The means of local alteration of Object Attributes is outside the scope of this standard.

4.5 Description de la conformité

4.5.1 Objet

L'objet du Conformance Block (bloc de conformité) est de permettre à différents environnements réels de communiquer à l'aide du même protocole DLMS malgré des capacités différentes. Il n'est pas obligatoire dans DLMS d'implémenter la totalité du service décrit. Seuls sont obligatoires les services Abort (abandon), Initiate (initialisation), GetStatus (obtenir le statut) et un GetNameList (obtenir la liste de noms) simplifié. Les autres services peuvent être négociés à l'aide du Conformance Block. Un service implémenté doit être totalement conforme à la spécification DLMS. Un service non obligatoire décrit dans le Conformance Block ne peut pas être utilisé par un client DMLS si le bit de présence n'est pas activé.

4.5.2 Structure

Tableau 2 - Description de la conformité

Description de la conformité	Code
Conformance	M
GetDataSetAttribute	M
GetTIAttribute	M
GetVariableAttribute	M
Read	M
Write	M
UnconfirmedWrite	M
ChangeScope	M
Start	M
Stop Resume	M
MakeUsable	M
DataSet Load	M
Selection in GetNameList	M
Detailed Access	M
Multiple Variable List	M
Data Set Upload	M

4.5.3 Paramètres

Le drapeau de l'attribut GetDataSet (obtenir le jeu données) indique si ce service est supporté.

Le drapeau de l'attribut GetTI (obtenir les TI - task invocation) indique s'il y a ou non au moins un objet TI défini dans le VDE. S'il y a au moins un objet TI, le service GetTIAttribute doit être supporté, pleinement implémenté et conforme à la description dans la présente norme. Il n'y a pas de conformité partielle possible pour ce service.

4.5 Conformance description

4.5.1 Purpose

The Conformance Block is proposed in order to allow various real environments to communicate using the same DLMS protocol through different capabilities. In DLMS, it is not mandatory to implement all the described services. Only the Initiate, Abort, GetStatus and a simple GetNameList services are mandatory. Other services may be negotiated using the Conformance Block. An implemented service shall be fully conform to its DLMS specification. A non-mandatory service described in the Conformance Block with a non-set bit may not be requested by a DLMS client.

4.5.2 Structure

Table 2 - Conformance description

Conformance description	Code
Conformance	M
GetDataSetAttribute	M
GetTIAttribute	M
GetVariableAttribute	M
Read	M
Write	M
UnconfirmedWrite	M
ChangeScope	M
Start	M
Stop Resume	M
MakeUsable	M
DataSet Load	M
Selection in GetNameList	M
Detailed Access	M
Multiple Variable List	M
Data Set Upload	M

4.5.3 Parameters

The GetDataSetAttribute flag describes whether or not the GetDataSetAttribute service is supported.

The GetTIAttribute flag describes whether or not at least one TI object is defined at the VDE. If at least one TI object exists, the GetTIAttribute service shall be supported, fully implemented and fully conform to its description in this standard. No partial conformance is allowed for this service.

Le drapeau de l'attribut GetVariable (obtenir une variable) indique s'il y a ou non, au moins un objet variable (un objet variable désigné, un objet liste de variables désignées ou un objet boîte à messages) défini dans le VDE. S'il y a au moins un objet variable, le service GetVariableAttribute doit être supporté, pleinement implémenté et conforme à sa description dans la présente norme. Il n'y a pas de conformité partielle possible pour ce service.

Les drapeaux Read (lecture), Write (écriture), UncomfirmedWrite (écriture non confirmée), ChangeScope (changement de champ), Start (démarrage) et MakeUsable (rendre disponible), indiquent si les services correspondants sont ou non disponibles dans ce VDE. Si un service est marqué comme étant disponible, il doit être pleinement implémenté et conforme à sa description dans la présente norme. Il n'y a pas de conformité partielle possible pour ces services.

Le drapeau Stop Resume (arrêt et reprise) indique que les services Stop (arrêt) et Resume (reprise) sont disponibles dans le VDE. Si ce drapeau est mis en place, les deux services doivent être pleinement implémentés et conformes à leurs descriptions dans la présente norme. Il n'y a pas de conformité partielle possible pour ces services.

Le drapeau DataSet Load (chargement de jeu de données) indique si les services InitiateLoad (initialiser le chargement), LoadSegment (chargement du segment) et TerminateLoad (fin chargement) sont disponibles dans le VDE. Si ce drapeau est positionné, les trois services doivent être pleinement implémentés et conformes à leurs descriptions dans la présente norme. Il n'y a pas de conformité partielle possible pour ces services.

Le drapeau Selection in GetNameList (recherche dans une liste de noms) indique si le service Selection in the GetNameList est supporté. Le service GetNameList doit toujours être supporté dans son expression la plus simple. C'est un GetNameList où il n'y a aucune demande de sélection particulière par le client. Le service GetNameList introduit un chemin plus sophistiqué pour atteindre une liste de noms d'objets dans VDE. Il est possible de sélectionner des noms par leurs durées de vie, leurs classes ou leurs champs d'accès. Cette extension au service GetNameList de base doit être supportée si le drapeau Selection in GetNameList est positionné.

Le descripteur Detailed Access (accès détaillé) spécifie sur deux bits le niveau de détails d'accès installé. La description s'établit comme suit :

- 00 : Pas d'Accès Détaillé supporté ;
- 01 : Accès Détaillé supporté sur un seul niveau ;
- 10 : Accès Détaillé supporté pour deux niveaux ;
- 11 : Accès Détaillé totalement supporté, quelle que soit la récursivité.

Le drapeau Multiple Variable List indique si l'implémentation est capable de gérer, dans les indications Read et Write et dans les requêtes d'InformationReport, des listes de variables composées de plus d'une variable. Si ce drapeau n'est pas positionné, le serveur a seulement la capacité de gérer des listes constituées en fait d'une seule variable de classe Variable Nommée ou Boîte à Messages.

Le drapeau Data Set Upload indique si les services de télérapatriement InitiateUpLoad, UpLoadSegment et TerminateUpLoad sont disponibles dans le VDE. Si ce drapeau est positionné, les trois services doivent être complètement réalisés et conformes à leur description dans la présente norme. Aucune conformité partielle n'est autorisée pour ces services.

The GetVariableAttribute flag describes whether or not at least one variable object (a Named Variable object, a Named Variable List object or a Message Box object) is defined at the VDE. If at least one variable object exists, the GetVariableAttribute service shall be supported, fully implemented and fully conform to its description in this present document. No partial conformance is allowed for this service.

The Read, Write, UnconfirmedWrite, ChangeScope, Start and MakeUsable flags describe whether or not the corresponding services are available at the VDE. If a service is described as available, it shall be fully implemented and fully conform to its description in this standard. No partial conformance is allowed for these services.

The Stop Resume flag describes whether or not Stop service and Resume service are available at the VDE. If this flag is set, both services shall be fully implemented and fully conform to their descriptions in this standard. No partial conformance is allowed for these services.

The DataSet Load flag specifies whether InitiateLoad service, LoadSegment service and TerminateLoad service are available at the VDE. If this flag is set, all three services shall be fully implemented and fully conform to their descriptions in this standard. No partial conformance is allowed for these services.

The Selection in GetNameList flag describes whether selection in the GetNameList service is supported. GetNameList service in its simplest presentation shall always be supported. That is a GetNameList where no special selection is requested by the client. The GetNameList service description introduces also a more sophisticated way to list object names defined at a VDE. It is possible to select names by their lifetime, by their object class or by their scope of access. This enhancement of the basic GetNameList service shall be supported if the Selection in GetNameList flag is set.

The Detailed Access descriptor specifies on two bits the supported nested level for detailed accesses. The description is as follows:

- 00: No Detailed Access is supported;
- 01: Detailed Access is supported for one single level;
- 10: Detailed Access is supported for two levels;
- 11: Detailed Access is fully supported, whatever may be the recursivity.

The Multiple Variable List flag indicates whether the implementation is capable of handling, in Read and Write indications and in InformationReport requests, lists of variables that are composed of more than one variable. If not set, the server may only be able to process lists consisting of one single variable of class Named Variable or Message Box.

The Data Set Upload flag specifies whether InitiateUpLoad service, UpLoadSegment service and TerminateUpLoad service are available at the VDE. If this flag is set, all three services shall be fully implemented and fully conform to their descriptions in this standard. No partial conformance is allowed for these services.

5 Services de gestion du contexte

5.1 Introduction

Les services de gestion du contexte sont les services Initiate (initialisation) et Abort (abandon).

Ces services permettent au client DLMS :

- de déclarer leurs caractéristiques de communication au serveur DLMS dans le contexte DLMS ; cela afin d'établir les besoins et les ressources supportant la communication ;

- d'interrompre une communication avec un serveur DLMS dans un contexte DLMS.

5.2 Le service Initiate (*initialisation)*

5.2.1 Objet

Le service Initiate sert à établir le contexte DLMS. Il permet aux utilisateurs communicants de DLMS d'échanger des informations concernant leurs besoins en ressources et en communications.

Le service Initiate fournit aussi les moyens d'identifier le client DLMS dans l'environnement DLMS par le biais de la création d'un objet VAA (association virtuelle d'applications). Cet objet VAA sert à définir le champ d'accès des objets de VDE relatifs à ce client DLMS.

Le service Initiate doit avoir fonctionné avec succès avant que tout autre service puisse être actionné.

5 Context management services

5.1 Introduction

The context management services are the Initiate service and the Abort service.

These services allow the DLMS client:

- to declare its communication characteristics to a DLMS server in the DLMS context. This is to establish the requirements and resources that will support that communication;

- to abort a communication with a DLMS server in the DLMS context.

5.2 Initiate service

5.2.1 Purpose

The initiate service is used to establish the DLMS context. It allows the communicating DLMS users to exchange information about their resources and communication requirements.

The Initiate service provides also the means to identify the DLMS client in the DLMS environment through the creation of a VAA object. This VAA object is used to define the scope of access of the VDE objects related to this DLMS client.

The Initiate service must be performed successfully before any other services can be operated.

5.2.2 Structure

La structure des composants des primitives du service est donnée ci-dessous.

Tableau 3 - Le service Initiate

Service initiate	Req	Ind	Resp	Conf
Argument	M	M(=)		
Dedicated Key	U	U(=)		
Response Allowed	U	U(=)		
Proposed Quality Of Service	U	U(=)		
Proposed DLMS Version Number	M	M(=)		
Proposed Conformance	M	M(=)		
Proposed Max Pdu Size	M	M(=)		
Result(+)			C	C(=)
Negotiated Quality Of Service			U	U(=)
Negotiated DLMS Version Number			M	M(=)
Negotiated Conformance			M	M(=)
Negotiated Max Pdu Size			M	M(=)
VAA Name			M	M(=)
Result(-)			C	C(=)
Error Type			M	M(=)

NOTES

1 L'ensemble de ces paramètres peut être contenu dans la partie données utilisateur de la demande A_Unit_Data d'ACSE. Dans ce cas, des paramètres supplémentaires sont requis pour complémenter la demande A_Unit_Data.

2 Dans le système d'appel, le prestataire DLMS peut diminuer la valeur des paramètres de la primitive request (demande). Dans le serveur DLMS, le prestataire DLMS peut réduire la valeur des paramètres de la primitive indication. Il n'y a pas d'autres modifications autorisées de ces valeurs.

3 Les primitives Result(+) ou Result(-) peuvent être transmises si et seulement si la valeur du paramètre Response Allowed est TRUE ou si ce paramètre est absent.

5.2.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques des demandes de service Initiate.

Le paramètre Dedicated Key (clé dédiée) contient une clé de chiffrement qui peut être utilisée au cours de transmissions ultérieures pour chiffrer des PDUs DLMS échangées entre les mêmes client et serveur. Son utilisation est permise seulement lorsque la PDU Initiate request (demande d'initialisation) à laquelle elle appartient a été encodée en utilisant une clé globale.

NOTE – Les manières de positionner la clé globale ou la clé dédiée ne sont pas du ressort de cette norme.

5.2.2 Structure

The structure of the component service primitives is shown below.

Table 3 - The Initiate service

Initiate service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Dedicated Key	U	U(=)		
Response Allowed	U	U(=)		
Proposed Quality Of Service	U	U(=)		
Proposed DLMS Version Number	M	M(=)		
Proposed Conformance	M	M(=)		
Proposed Max Pdu Size	M	M(=)		
Result(+)			C	C(=)
Negotiated Quality Of Service			U	U(=)
Negotiated DLMS Version Number			M	M(=)
Negotiated Conformance			M	M(=)
Negotiated Max Pdu Size			M	M(=)
VAA Name			M	M(=)
Result(-)			C	C(=)
Error Type			M	M(=)

NOTES

1 The collection of these parameters may be contained in the user data portion of the A_Unit_Data request of the ACSE. In this case, additional parameters are required to complete the A_Unit_Data request.

2 The DLMS provider in the calling system may reduce the values of the parameters in the request primitive. The DLMS provider in the DLMS server may reduce the values of the parameters in the indication primitive. No other modification is permitted to these values.

3 The Result(+) or Result(-) primitives may be transmitted if and only if the value of the Response Allowed parameter is TRUE or this parameter is absent.

5.2.3 Parameters

The Argument parameter conveys the specific parameters of the Initiate Service request.

The Dedicated Key parameter contains a ciphering key that may be used in subsequent transmissions to cipher DLMS PDUs exchanged between the same client and server. Its use is allowed only when the Initiate request PDU to which it belongs has been encoded using a Global Key.

NOTE – The ways of setting the Global Key or the Dedicated Key are outside the scope of this standard.

Le paramètre Response Allowed (réponse autorisée) indique que le service Initiate est confirmé (si TRUE) ou non confirmé (si FALSE). Dans le premier cas, une réponse par le serveur DLMS est obligatoire. Dans le deuxième cas, elle est interdite.

NOTE – Le propos de ce paramètre est de rendre possible la diffusion des demandes d'initialisation (Initiate requests).

Le paramètre Proposed Quality Of Service (qualité de service proposée) permet au client de demander un niveau spécifique de qualité de service du serveur. La sémantique exacte de ce paramètre n'est pas du ressort de cette norme. Elle peut être fixée par des spécifications d'accompagnement ou par un accord externe entre client et serveur.

Le paramètre Proposed DLMS Version Number (numéro de version de DLMS proposé), du type Unsigned8 (nombre non signé sur 8 bits), contient la version la plus récente supportée par le client DLMS.

Le paramètre Proposed Conformance (Conformité Proposée), de type BIT STRING (chaîne binaire), définit la liste des services et des fonctionnalités supportées par le client DLMS. L'objet de ce paramètre est de permettre au prestataire DLMS d'allouer les ressources appropriées afin de garantir la communication. Le contenu exact du Conformance Block (bloc de conformité) est décrit en 4.5.

Le paramètre Proposed Max PDU Size (taille maximale proposée du PDU), de type Unsigned16 (nombre non signé sur 16 bits), propose une longueur maximale exprimée en octets, des PDU de DLMS échangés. La valeur proposée dans une demande Initiate doit être suffisante pour toujours permettre la transmission du PDU "Initiate Error" (erreur d'initialisation).

Le paramètre Result(+) (résultat positif) indique que le service demandé a fonctionné.

Le paramètre Quality Of Service est le paramètre permettant au serveur de garantir au client un certain niveau de service. La sémantique exacte de ce paramètre n'est pas du ressort de cette norme. Elle peut être fixée par des spécifications d'accompagnement ou par un accord externe entre client et serveur.

Le paramètre Negotiated DLMS Version Number (numéro de version de DLMS négocié), de type Unsigned8, spécifie la version de DLMS supportée par le VDE-handler (gestionnaire VDE). La valeur de ce paramètre doit être inférieure ou égale à celle du paramètre Proposed DLMS Version Number.

Le paramètre Negotiated Conformance (conformité négociée), de type BIT STRING (chaîne binaire), définit la liste des services et fonctionnalités utilisables par la suite. Ce paramètre est traité comme un "et" logique entre la conformité proposée et la conformité interne du VDE-handler.

Le paramètre Negotiated Max PDU Size (Taille maximale négociée du PDU), de type Unsigned16, contient la longueur maximale exprimée en octets, pour les PDU de DLMS échangés. Une PDU qui serait plus longue que cette longueur maximale ne serait pas considérée. La longueur maximale est traitée comme le minimum de Proposed Max PDU Size et le maximum de la taille de PDU que le VDE-handler peut supporter.

The Response Allowed parameter indicates that the Initiate service is confirmed (if TRUE) or unconfirmed (if FALSE). In the first case, a response by the DLMS server is mandatory. In the second case, it is forbidden.

NOTE – The intended purpose of this parameter is to make possible the broadcasting of Initiate requests.

The Proposed Quality Of Service parameter enables the client to request a specific grade of service from the server. The exact semantics of this parameter is outside the scope of this standard. It may be fixed by companion specifications or by an external agreement between the client and the server.

The Proposed DLMS Version Number parameter, of type Unsigned8, contains the highest DLMS version level supported by the DLMS client.

The Proposed Conformance parameter, of type BIT STRING, defines the list of the services and functionalities that are supported by the DLMS client. The purpose of this parameter is to allow the DLMS provider to allocate the appropriate resources to ensure communication. The exact content of the Conformance Block is described in 4.5.

The Proposed Max Pdu Size parameter, of type Unsigned16, proposes a maximum length expressed in bytes for the exchanged DLMS PDUs. The value proposed in an Initiate request must be large enough to always permit the Initiate Error PDU transmission.

The Result(+) parameter indicates that the requested service has succeeded.

The Negotiated Quality Of Service parameter enables the server to grant a specific grade of service to the client. The exact semantics of this parameter is outside of the scope of this standard. It may be fixed by companion specifications or by an external agreement between the client and the server.

The Negotiated DLMS Version Number parameter, of type Unsigned8, specifies the DLMS version level supported by the VDE-handler. The value of the parameter must be lower than or equal to the Proposed DLMS Version Number parameter.

The Negotiated Conformance parameter, of type BIT STRING, defines the list of the services and functionalities that may be used afterwards. This parameter is computed as a logical "and" between the submitted conformance and the VDE-handler internal conformance.

The Negotiated Max PDU Size parameter, of type Unsigned16, contains a maximum length expressed in bytes for the exchanged DLMS PDUs. A PDU that is longer than this maximum length will be discarded. This maximum length is computed as the minimum of the Proposed Max Pdu Size and the maximum PDU size that the VDE-handler may support.

Le paramètre VAA Name (nom de VAA), de type Object Name (nom d'objet), n'identifie, dans le VDE, que l'objet VAA associé au client DLMS. On peut, à tout moment par la suite, se référer à cet objet, en particulier pour vérifier la validité de l'accès aux objets de DLMS.

Le paramètre Result(-) (résultat négatif) indique que le service demandé auparavant a échoué. Le paramètre Error Type (type d'erreur) fournit la raison du défaut. Le paramètre Error Type est décrit en détails en A.5.

5.2.4 Procédure de service

La demande de service Initiate peut être faite à n'importe quel moment par le client DLMS.

A la réception d'une primitive indication du service Initiate, le serveur DLMS vérifie les paramètres fournis ainsi que l'existence préalable de l'objet VAA associé. Aucune réponse n'est émise si le paramètre Response Allowed (réponse autorisée) dans la primitive demande Initiate est présent et positionné à la valeur FALSE.

Un résultat négatif est émis dans une primitive réponse de Initiate à l'aide du code d'erreur approprié lorsque l'un des cas suivants se présente :

- la version de DLMS proposée est inférieure à celle qui est supportée ;

- la liste des services supportés n'est pas suffisante ;

- la taille maximale du PDU n'est pas suffisante ;

- il est nécessaire de créer une nouvelle VAA et le VDE-handler ne peut pas le faire.

Dans tous les autres cas, la réponse doit être positive. Si l'objet VAA associé n'existe pas déjà, il est créé. S'il existe, l'objet existant est réutilisé.

NOTES

1 Une spécification d'accompagnement peut définir les conditions à satisfaire pour exécuter avec succès un service Initiate.

2 Si une PDU dépasse la taille maximale négociée, cette PDU ne sera pas considérée; un "Abort" peut être généré.

3 VDE-handler doit avoir un comportement déterministe. Quand il reçoit deux primitives indication de Initiate avec les mêmes paramètres et se référant à une même VAA déjà existante, il doit renvoyer exactement le même PDU de réponse.

The VAA Name parameter, of type ObjectName, identifies uniquely within the VDE, the VAA object associated with the DLMS client. Reference to this object may be made at any further time, especially to check the validity of an access to DLMS objects.

The Result(-) parameter indicates that the service previously requested has failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in details in A.5.

5.2.4 Service procedure

The Initiate service request may be issued at any time by the DLMS client.

On receipt of an Initiate indication service primitive, the DLMS server checks the provided parameters and the existence of an already existing associated VAA object. No response is issued if the Response Allowed parameter in the Initiate request primitive was present and set to FALSE.

A negative result is issued in an Initiate response primitive with the appropriate error code if one of the following statements is met:

- the proposed DLMS version number is lower than the supported one;
- the list of supported services is not sufficient;
- the maximum PDU size is not sufficient;
- a new VAA should be created but the VDE-handler cannot do it.

Otherwise, the response shall be successful. If the associated VAA object does not exist already, the VAA object is created. If it exists, the existing VAA object is reused.

NOTES

1 Companion specification may define conditions that are to be met to successfully perform an Initiate service.

2 If a PDU exceeds the negotiated PDU size, the PDU will be discarded; an Abort may be generated.

3 The VDE-handler shall have a deterministic behaviour. When receiving two Initiate indication primitives with the same parameters and referring to a common existing VAA, it shall return exactly the same response PDU.

5.3 Le service Abort *(abandon)*

5.3.1 Objet

Le service Abort sert à abandonner brusquement et sans négociation un contexte DLMS. Le client DLMS émet une primitive de demande Abort pour indiquer qu'il désire, immédiatement et sans négociation, interrompre une communication dans un contexte DLMS.

5.3.2 Structure

Tableau 4 - Le service Abort

Service abort	Req	Ind
Argument	M	M(=)

5.3.3 Paramètres

Le paramètre Argument véhicule une valeur NULL, car il n'y a pas de paramètres transmis au service Abort.

5.3.4 Procédure de service

La demande de service Abort peut être émise à n'importe quel moment par le client DLMS.

Le serveur DLMS tout d'abord vérifie la valeur de l'attribut Abortable de la VAA associée. Si elle est égale à VRAI, le contexte DLMS est effacé et l'objet VAA détruit. Si le Data Set était dans l'état Loading et dans le champ d'accès du VAA, alors l'état du Data Set est positionné à Empty (vide).

NOTE – Cela correspond à effacer les fragments du Data Set qui auraient pu être déjà chargés. Tous les objets définis dans le champ d'accès VAA-specific voient leur champ d'accès devenir VDE-spécific, y compris les objets jeu de données et TI si le VAA est l' Executive VAA.

Si l'attribut Abortable de la VAA associée est positionné à FAUX, alors rien ne se passe.

Avant tout autre utilisation de services DLMS, le client DLMS doit réitérer avec succès un nouveau service Initiate.

5.3 Abort service

5.3.1 Purpose

The Abort service is used to relinquish the DLMS context abruptly, without negotiation. The DLMS client issues the Abort request primitive to indicate that it wishes to immediately, and without negotiation, discontinue a communication in the DLMS context.

5.3.2 Structure

Table 4 - The Abort service

Abort service	Req	Ind
Argument	M	M(=)

5.3.3 Parameter

The Argument parameter conveys a NULL value because no parameter is transmitted with the Abort service.

5.3.4 Service procedure

The Abort service request may be issued at any time by the DLMS client.

The DLMS server first checks the Abortable attribute of the associated VAA. If the Abortable attribute of the VAA is set to TRUE, the DLMS context is cancelled and the VAA object is deleted. If the Data Set was in the Loading state and within the scope of the VAA, then the state of this Data Set is set to Empty.

NOTE – This corresponds to deleting the fragments of the Data Set that may already have been loaded. All the objects defined with this VAA-specific scope of access have their scope changed to VDE-specific including both the Data Set and the TI objects if the VAA was the Executive VAA.

If the Abortable attribute of the VAA is set to FALSE, then nothing happens.

Before any further use of DLMS services, the DLMS client must perform again a successful Initiate service.

6 Services de support du VDE

L'Equipement Virtuel de Distribution (VDE : Virtual Distribution Equipement) est défini plus haut. Le présent article décrit les structures des objets VDE et les services associés. Les services de support du VDE sont les services GetStatus (obtenir l'état) et GetNameList (obtenir la liste des noms).

6.1 Equipement virtuel de distribution

6.1.1 Les objets VDE

Le VDE représente la part d'un processus d'application de DLMS (DLMS AP) dont le comportement est modélisé sous la forme d'une seule entité (voir article 4). Le VDE est la principale entité abstraite de la spécification DLMS.

6.1.1.1 Les attributs

Les attributs du VDE sont :

Object VDE	
key attribute : VDE-handler	(statique)
attribute : VDE type	(statique)
attribute : Serial Number	(statique)
attribute : Vendor Name	(statique)
attribute : Model	(statique)
attribute : Version Number	(statique)
attribute : Resources	(statique)
attribute : List of VAA	(dynamique)
attribute : Status	(dynamique)

Les attributs statiques sont préconfigurés dans le VDE. Ce sont les attributs VDE Type, Serial Number (numéro de série), Vendor Name (nom du fabricant), Model, Version Number et Ressources. Les attributs statiques ne sont ni chargeables ni inscriptibles. Ils identifient les caractéristiques constructeur de l'installation en place. Les caractéristiques spécifiées dans ces attributs le sont à des fins de gestion.

Les autres attributs sont dynamiques. Ils décrivent la configuration du VDE.

6 VDE support services

The Virtual Distribution Equipment is defined above. This clause describes the structure of the VDE object and the associated services. The VDE support services are the GetStatus and the GetNameList services.

6.1 Virtual distribution equipment description

6.1.1 The VDE object

The VDE represents the part of a DLMS AP whose behaviour is modelled as a single entity (see clause 4). The VDE is the main abstract entity of the DLMS specification.

6.1.1.1 Attributes

The attributes of the VDE are:

Object: VDE

key attribute: VDE-handler	(static)
attribute: VDE Type	(static)
attribute: Serial Number	(static)
attribute: Vendor Name	(static)
attribute: Model	(static)
attribute: Version Number	(static)
attribute: Resources	(static)
attribute: List of VAA	(dynamic)
attribute: Status	(dynamic)

The static attributes are pre-configured within a VDE. They are the VDE Type, the Serial Number, the Vendor Name, the Model, the Version Number and the Resources attributes. The static attributes are non-loadable and non-writable. They identify the vendor characteristics of the current implementation. The characteristics specified in these attributes are for revision management purposes.

The other attributes are dynamic. They describe the configuration of the VDE.

6.1.1.2 Description

L'attribut VDE-handler (gestionnaire VDE) identifie le VDE de manière unique parce que l'existence d'un VDE-handler totalement fonctionnel coïncide exactement avec l'existence du VDE.

L'attribut VDE Type (type de VDE) spécifie le type du VDE. Ce paramètre identifie de façon unique le type du VDE dans le monde. L'attribution d'un type à un VDE est en dehors du domaine de cette spécification.

L'attribut Serial Number (numéro de série) identifie le VDE de façon unique dans le monde.

L'attribut Vendor Name (nom du fabricant) identifie le nom du fabricant du système qui supporte le VDE.

L'attribut Model (modèle) identifie le modèle du système qui supporte le VDE.

L'attribut Version Number (numéro de version) identifie le niveau de version de logiciel de communication du système qui supporte le VDE.

L'attribut Status (état) indique l'état logique du VDE. Trois états sont autorisés: READY (prêt), NOCHANGE (pas de modification) et INOPERABLE (indisponible). L'état normal du VDE est READY, tandis que les deux autres états sont transitoires. Dans l'état READY, tous les services sont disponibles. Dans l'état NOCHANGE, les seuls services disponibles sont ceux qui ne modifient pas les attributs des objets définis dans le VDE, c'est à dire: GetStatus, Read, InformationReport, GetNameList, GetDataSetAttributes, les trois services du télérapatriement, GetTIAttribute et GetVariableAttribute. Dans l'état INOPERABLE, aucune des fonctions de l'équipement réel représenté par le VDE n'est opérationnelle.

L'attribut Resources décrit les ressources physiques de l'équipement réel de distribution représenté par le VDE.

La liste des attributs de VAA identifie les noms des objets VAA qui sont généralement définis dans le VDE. Les objets VAA sont décrits à l'article 8.

6.1.2 Contenu de VDE

Du point de vue de DLMS, un VDE contient zéro ou un jeu de données, zéro, une ou plusieurs TI, zéro, une ou plusieurs variables.

L'objet variable défini dans le VDE peut être une variable désignée, une liste de variables désignées ou une boîte à messages.

6.1.1.2 Description

The VDE-handler attribute identifies uniquely the VDE because the existence of a fully functional VDE-handler corresponds exactly with the existence of the VDE.

The VDE Type attribute specifies the type of the VDE. This parameter identifies uniquely the type of the VDE throughout the world. The attribution of a type to a VDE is out of the scope of this specification.

The Serial Number attribute identifies uniquely the VDE throughout the world.

The Vendor Name attribute identifies the manufacturer of the system which supports the VDE.

The Model attribute identifies the model of the system which supports the VDE.

The Version Number attribute identifies the software revision level of the communication system which supports the VDE.

The Status attribute indicates the logical status of the VDE. Three states are allowed: READY, NOCHANGE and INOPERABLE. The normal state of the VDE is the READY state, while the two other states are transient. In the READY state, all services are allowed. In the NOCHANGE state, the only services allowed are the services which do not modify the attributes of an object defined at the VDE, namely GetStatus, Read, InformationReport, GetNameList, GetDataSetAttributes, the three UpLoad services, GetTIAttribute and GetVariableAttribute. In the INOPERABLE state, none of the functions of the real device represented by the VDE is operational.

The Resources attribute describes the physical resources of the real distribution device represented by the VDE.

The List of VAA attribute identifies the VAA objects names that are currently defined within the VDE. The VAA object is described in clause 8.

6.1.2 VDE Content

From the DLMS point of view, the VDE contains zero or one Data Set, zero or more TIs and zero or more variables.

The variable object defined at the VDE may be Named Variable, Named Variable List or Message Box.

6.1.3 Opérations sur le VDE

Les services Getstatus et GetNameList fonctionnent dans le VDE.

Ces services permettent au client DLMS :

- d'obtenir l'état d'un VDE, ses ressources, la liste des objets VAA définis et, éventuellement, les caractéristiques spécifiques de la fabrication du système ;

- d'obtenir la liste des objets définis dans le VDE.

6.2 Le service GetStatus (*obtenir l'état)*

6.2.1 Objet

L'objet de GetStatus est de déterminer l'état général du VDE interrogé; il peut aussi identifier les attributs spécifiques du fabricant.

6.2.2 Structure

Tableau 5 - Le service GetStatus

Service GetStatus	Req	Ind	Resp	Conf
Argument	M	M(=)		
Identify	M	M(=)		
Result(+)			S	S(=)
VDE Type			M	M(=)
Serial Number			M	M(=)
Status			M	M(=)
List of VAA			M	M(=)
Identify			U	U(=)
Resources			C	C(=)
Vendor Name			C	C(=)
Model			C	C(=)
Version Number			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

6.2.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service GetStatus.

Le paramètre Identify (identifier) indique à l'utilisateur qui répond s'il doit inclure tous les attributs statiques du VDE dans la primitive réponse de GetStatus. Si la valeur du paramètre est TRUE (vrai), le VDE qui répond indique toutes ses caractéristiques statiques dans la primitive réponse de GetStatus.

Le paramètre Result(+) indique que le service demandé a fonctionné.

6.1.3 Operations on the VDE

The GetStatus and the GetNameList services operate on the VDE.

These services allow the DLMS client:

- to obtain the status of the VDE, its resources, the list of the defined VAA objects and, eventually, the specific characteristics of the manufactured system;

- to obtain the list of the objects defined at the VDE.

6.2 GetStatus service

6.2.1 Purpose

The purpose of the GetStatus service is to determine the general condition of a responding VDE. It may also identify the vendor specific attributes.

6.2.2 Structure

Table 5 - The GetStatus service

GetStatus service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Identify	M	M(=)		
Result(+)			S	S(=)
VDE Type			M	M(=)
Serial Number			M	M(=)
Status			M	M(=)
List of VAA			M	M(=)
Identify			U	U(=)
Resources			C	C(=)
Vendor Name			C	C(=)
Model			C	C(=)
Version Number			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

6.2.3 Parameters

The Argument parameter conveys the specific parameter of the GetStatus service request.

The Identify parameter indicates to the responding user if it must include all the static attributes of the VDE in the GetStatus response primitive. If the parameter value is set to TRUE, the responding VDE indicates all its static characteristics in the GetStatus response primitive.

The Result(+) parameter indicates that the requested service has succeeded.

Le paramètre VDE Type, de type Integer16, spécifie le type du VDE. Ce paramètre identifie de façon unique le type du VDE dans le monde. L'attribution d'un type à un VDE n'est pas dans le domaine de cette spécification.

Le paramètre Serial Number, de type chaîne d'octets, identifie le VDE de façon unique dans le monde.

Le paramètre Status, de type ENUMERATED (énumération), indique l'état logique du VDE. Trois états sont permis: READY, NOCHANGE et INOPERABLE. Dans l'état READY, tous les services sont disponibles. Dans l'état NOCHANGE, les seuls services disponibles sont ceux qui ne modifient pas les attributs des objets VDE. Dans l'état INOPERABLE, aucune des fonctions de l'équipement réel représenté par le VDE n'est disponible.

Le paramètre List of VAA, de type SEQUENCE (séquence) de noms d'objets, identifie les objets VAA généralement définis dans le VDE.

Le paramètre Identify, de type BOOLEAN (booléen), indique si tous les attributs statiques du VDE sont inclus dans la primitive réponse du GetStatus. Si la valeur du paramètre est TRUE (vrai), le VDE qui répond indique toutes ses caractéristiques statiques.

Les paramètres suivants sont inclus dans une primitive réponse de GetStatus seulement si la valeur du paramètre Identify est TRUE dans la primitive Indication.

Le paramètre Resources, de type VisibleString (chaîne en clair), décrit les ressources physiques de l'équipement de distribution réel que décrit le VDE.

Le paramètre Vendor Name (nom du fabricant), de type VisibleString, identifie le fabricant du système qui supporte le VDE. La valeur du paramètre est fixée par le fabricant.

Le paramètre Model, de type VisibleString, identifie le modèle du système qui supporte le VDE. La valeur du paramètre est fixée par le fabricant.

Le paramètre Version Number (numéro de version), de type Unsigned8, identifie le niveau de version du logiciel de communication du système qui supporte le VDE. La valeur du paramètre est fixée par le fabricant.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit les raisons de l'échec. Le paramètre Error Type est décrit en détails en A.5.

6.2.4 Procédure de service

Le serveur DLMS réalise le service GetStatus en déterminant l'information obligatoire pour la création d'une réponse valide. Cette information est directement subordonnée à l'objet VDE.

The VDE Type parameter, of type Integer16, specifies the type of the VDE. This parameter identifies uniquely the type of the VDE throughout the world. The attribution of a type to a VDE is outside the scope of this specification.

The Serial Number parameter, of type Octet String, identifies uniquely the VDE throughout the world.

The Status parameter, of type ENUMERATED, indicates the logical status of the VDE. Three states are allowed: READY, NOCHANGE and INOPERABLE. In the READY state, all services are allowed. In the NOCHANGE state, the only services allowed are the services which do not modify the attributes of an VDE object. In the INOPERABLE state, none of the functions of the real device represented by the VDE is operational.

The List of VAA parameter, of type SEQUENCE OF ObjectName, identifies the VAA objects that are currently defined within the VDE.

The Identify parameter, of type BOOLEAN, indicates if all the static attributes of the VDE are included in the GetStatus response primitive. If the parameter value is set to TRUE, the responding VDE indicates all its static characteristics.

The following parameters are included in the GetStatus response primitive only if the Identify parameter was set to TRUE in the indication primitive.

The Resources parameter, of type VisibleString, describes the physical resources of the real distribution device represented by the VDE.

The Vendor Name parameter, of type VisibleString, identifies the manufacturer of the system which supports the VDE. The value of the parameter is assigned by the manufacturer.

The Model parameter, of type VisibleString, identifies the model of the system which supports the VDE. The value of the parameter is assigned by the manufacturer.

The Version Number parameter, of type Unsigned8, identifies the software revision level of the communication system which supports the VDE. The value of the parameter is assigned by the manufacturer.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in details in A.5.

6.2.4 Service procedure

The DLMS server performs the GetStatus service by determining the information mandatory to create a valid response. This information is directly subordinated to the VDE object.

6.3 Le service GetNameList *(obtenir une liste de noms)*

6.3.1 Objet

Le service GetNameList est utilisé par les clients DLMS pour demander qu'un serveur DLMS renvoie la liste, ou une partie de la liste, des noms d'objets définis dans le champ d'action du VDE.

6.3.2 Structure

Tableau 6 - Le service GetNameList

Service GetNameList	Req	Ind	Resp	Conf
Argument	M	M(=)		
Lifetime Selection	U	U(=)		
Object Class Selection	U	U(=)		
Scope of Access Selection	U	U(=)		
VAA Name	U	U(=)		
Continue After	U	U(=)		
Result(+)			S	S(=)
More Follows			M	M(=)
List of Object Name			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

Le service GetNameList précise les caractéristiques des objets qu'il faut énumérer. La réponse contient la liste des noms d'objets qui satisfont la demande, peut être avec la mention "More Follows" (il y a une suite), ou un type d'erreur en cas d'insuccès.

6.3 GetNameList service

6.3.1 Purpose

The GetNameList service is used by a DLMS client in order to request that a DLMS server returns the list of, or part of the list of, object names defined within the scope of the VDE.

6.3.2 Structure

Table 6 - The GetNameList service

GetNameList service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Lifetime Selection	U	U(=)		
Object Class Selection	U	U(=)		
Scope of Access Selection	U	U(=)		
VAA Name	U	U(=)		
Continue After	U	U(=)		
Result(+)			S	S(=)
More Follows			M	M(=)
List of Object Name			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

The GetNameList service defines the characteristics of the objects that must be listed. The response contains the list of the object names that match the requirements possibly with "More Follows" or the error type if it fails.

6.3.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service GetNameList. Ce paramètre définit différents choix de configurations. Les objets peuvent être sélectionnés indépendamment par leurs durées de vie, leurs classes ou leurs champs d'accès.

Le paramètre optionnel Lifetime Selection (sélection par durée de vie), de type ENUMERATED (énumération), précise la durée de vie des objets qu'il faut énumérer. S'il est présent, ses valeurs sont VDE-AND-DATA-SET (VDE et jeu de données), ou VDE-ONLY (VDE seulement), ou DATA-SET-ONLY (jeu de données seulement). Si VDE-AND-DATA-SET est retenue, on n'émet pas de restrictions à la durée de vie et tous les objets définis au niveau VDE peuvent être énumérés. Si VDE-ONLY est retenue, seulement les objets définis au niveau VDE et pas dans le jeu de données peuvent être énumérés (durée de vie = VDE). Si DATA-SET-ONLY est retenue, seuls les objets définis dans le jeu de données (durée de vie = Data-Set) peuvent être énumérés. L'absence de ce paramètre est équivalente au choix VDE-AND-DATA-SET, ce qui veut dire qu'il n'y a pas de sélection sur la durée de vie des objets, et que tous les objets peuvent être énumérés.

Le paramètre optionnel Object Class Selection (sélection par type), de type ENUMARATED, spécifie quelle classe d'objets est demandée. Cette classe est l'une des suivantes : ALL pour tous les objets, NAMED-VARIABLES pour les variables désignées, NAMED-VARIABLE-LIST pour les listes de variables désignées, MESSAGE-BOX pour les boîtes à messages, TASK-INVOCATION pour les objets TI, DATA-SET pour les objets du jeu de données ou VAA pour les objets VAA. L'absence de ce paramètre est équivalente au choix ALL.

Le paramètre Scope of Access Selection (sélection par champ d'accès), de type ScopeOfAccess, spécifie le champ d'accès des objets à sélectionner. Il doit être spécifique VDE ou spécifique VAA. L'absence de ce paramètre signifie qu'il n'y a pas de sélection par rapport aux champs d'accès des objets: c'est-à-dire que tous les objets peuvent être énumérés.

Quand il est présent, le paramètre optionnel VAA Name, de type ObjectName (nom d'objet), spécifie le nom de la VAA associée. Ce paramètre est utile lorsque le paramètre Scope Of Access est positionné à VAA-specific.

Quand il est présent, le paramètre optionnel Continue After, de type ObjectName (nom d'objet), spécifie que le client DLMS souhaite que la liste des paramètres nom d'objet renvoyée par le serveur DLMS commence par un autre nom que le premier dans la liste. L'absence de ce paramètre signifie que la liste doit être retournée à partir du premier nom. Ce dispositif est surtout utilisé pour compléter une demande GetNameList dont la réponse contenait un paramètre More Follows (il y a une suite) dont la valeur est TRUE (vrai). Donner comme valeur de ce paramètre un nom d'objet (Object Name) qui est inconnu du serveur DLMS induira en retour l'émission de Result(-) avec un code d'erreur égal à OBJECT-UNDEFINED (objet non défini).

Le paramètre Result(+) indique que le service demandé a fonctionné.

Le paramètre List of Object Name, de type SEQUENCE de noms d'objets, contient tous les noms qui satisfont aux besoins de la durée de vie, la classe des objets et du champ d'accès décrits dans la primitive demande.

6.3.3 Parameters

The Argument parameter conveys the specific parameters of the GetNameList service request. The parameters define various selection configurations. The objects may be selected independently by their lifetime, their class or their scope of access.

The Lifetime Selection optional parameter, of type ENUMERATED, defines the lifetime of the objects that must be listed. If present, its value must be VDE-AND-DATA-SET, VDE-ONLY or DATA-SET-ONLY. If VDE-AND-DATA-SET is selected, no restriction on the lifetime is set and all the objects defined at the VDE may be listed. If VDE-ONLY is selected, only the objects that are defined at the VDE but not in the data set (lifetime = VDE) may be listed. If DATA-SET-ONLY is selected, only the objects that are defined at the VDE within the data set (lifetime = data set) may be listed. The absence of this parameter is equivalent to the choice VDE-AND-DATA-SET, which means that no selection is performed on the basis of the lifetime of the objects, and all objects may be listed.

The Object Class Selection optional parameter, of type ENUMERATED, specifies which class of objects is requested. This class is one of the following: ALL for all the objects, NAMED-VARIABLE for the Named Variable objects, NAMED-VARIABLE-LIST for the Named Variable List objects, MESSAGE-BOX for the Message Box objects, TASK-INVOCATION for the TI objects, DATA-SET for the data set objects or VAA for the VAA objects. The absence of this parameter is equivalent to the choice ALL.

The Scope of Access Selection parameter, of type ScopeOfAccess, specifies the scope of access of the objects to be selected. It must be VDE-specific or VAA-specific. The absence of this parameter means that no selection is performed on the basis of the Scope of Access of the objects: that is, all objects may be listed.

When present, the VAA Name optional parameter, of type ObjectName, specifies the associated VAA-Name. This parameter is useful when the Scope of Access Selection parameter is set to VAA-specific.

When present, the Continue After optional parameter, of type ObjectName, specifies that the DLMS client wishes that the list of Object Name parameter returned by the DLMS server begins with another name than the first in the list. The absence of this parameter means that the list must be returned beginning with the first name. This feature is mainly used to continue a previously GetNameList request whose response has contained a More Follows parameter set to TRUE. Setting the value of this parameter to an Object Name which is unknown for the DLMS server shall result in a Result(-) being sent back, with an error code equal to OBJECT-UNDEFINED.

The Result(+) parameter indicates that the requested service has succeeded.

The List of Object Name parameter, of type SEQUENCE OF ObjectName, contains all the names that match the requirements of lifetime, object class and scope of access described in the request primitive.

Le paramètre More Follows, de type BOOLEAN (booléen), s'il a la valeur TRUE (vrai) indique qu'il faut émettre une autre demande GetNameList pour obtenir davantage d'information. S'il prend la valeur FALSE (faux), alors le dernier nom est dans la liste des noms, ou la liste des noms d'objets est vide.

Le paramètre Result(-) indique que le service demandé a échoué. Le paramètre Error Type donne la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

6.3.4 Procédure de service

Le serveur DLMS renvoie la liste des noms qui satisfont durée de vie, classe d'objets et champ d'accès spécifiés dans le paramètre de la primitive indication. Si le paramètre Continue Choice est à FALSE (faux) dans la primitive indication, la liste commence avec le premier objet de liste satisfaisant aux conditions. Autrement, la liste renvoyée commence avec le premier nom suivant la valeur du paramètre Continue After et satisfaisant les paramètres de sélection : conceptuellement, tous les noms d'objet du VDE sont triés (le critère de tri est non pertinent tant qu'il assure un ordre total des noms d'objets) ; puis, tous les noms d'objet avant le nom d'objet qui apparaît dans le paramètre Continue After, et incluant celui-ci, sont écartés; enfin, la sélection est réalisée sur les noms d'objet restants afin d'obtenir la liste désirée.

6.3.5 Conformité

Le service GetNameList a une description particulière de la conformité. Une forme minimale du service doit toujours être supportée par le VDE. Cette forme minimale est équivalente à la demande de service sans sélection : Durée de vie = VDE ou jeu de données, Object Class = All, Champ d'accès absent (par exemple VDE-specific ou VAA-specific).

Si les paramètres de toutes les sélections sont supportés par le service GetNameList, le bit de sélection peut être mis en place dans le Conformance Block. Si certains ne sont pas supportés, le bit de Sélection dans GetNameList ne doit pas être positionné, et aucun paramètre de sélection ne doit apparaitre dans GetNameList.

The More Follows parameter, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) additional GetNameList requests are necessary to retrieve more of the requested information. If true, more requests are necessary. If false, then either the last name of the list is in the list of object name or the list of object name is empty.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in details in A.5.

6.3.4 Service procedure

The DLMS server returns the list of the names that match the lifetime, object class and scope of access specified in the parameters of the indication primitive. If the Continue Choice parameter is set to FALSE in the indication primitive, the list begins with the first name in the list of objects satisfying the conditions at the DLMS server. Otherwise, the list returned begins with the first object name following the Continue After value and satisfying the selection parameters: conceptually, all Object Names in the VDE are sorted (the criterion used for this sort is irrelevant as long as it provides a total ordering on Object Names); then, all Object Names before and including the Object Name that appears in the Continue After parameter are discarded; finally, the selection process is performed on the remaining Object Names to obtain the desired list.

6.3.5 Conformance

The GetNameList service has a special conformance description. A minimal form of the GetNameList service shall always be supported by a VDE. This minimal form corresponds to the service requested without selection: lifetime = VDE or Data Set, Object Class = All, Scope of Access absent (that is VDE- or VAA-specific).

If all selection parameters are supported in the GetNameList service, the Selection in GetNameList bit may be set in the conformance block. If some of them are not supported, the Selection in GetNameList bit shall not be set, and no selection parameter may appear in the GetNameList.

7 Services de gestion du jeu de données

Le modèle DLMS de l'équipement virtuel de distribution décrit ci-dessus introduit des éléments abstraits. Le présent article décrit les structures de l'objet jeu de données et les services qui gèrent le jeu de données.

Le jeu de données peut être chargé ou prédéfini. Les services fournis permettent à un client DLMS de manipuler le jeu de données défini dans le serveur DLMS.

7.1 Description du jeu de données

7.1.1 L'objet jeu de données

Le jeu de donnés représente un ensemble des capacités du VDE, utilisé dans un but spécifique.

7.1.1.1 Attributs

Les attributs du jeu de données sont :

Object : Data Set
- Key attribute : Data Set Name
- attribute : Scope Of Access (VDE-specific, VAA-specific)
- constraint scope of access = VAA-specific
  - attribute : Executive VAA Name
- attribute : Scope May Change (TRUE, FALSE)
- attribute : Data Set Content
- attribute : Loadable (TRUE, FALSE)
- attribute : List of Task Invocation
- attribute : State (EMPTY, LOADING, READY)
- constraint : State = LOADING
  - attribute : Current Segment
  - attribute : Number of Segments
- constraint: State=READY
  - attribute : UpLoading (TRUE, FALSE)

L'objet jeu de données est toujours prédéfini si sa durée de vie est toujours VDE. Si l'Executive VAA existe dans le VDE, le domaine d'accès de l'objet jeu de données est toujours VAA-specific, et l'attribut nom spécifie le nom de l'Executive VAA.

7 Data set management services

The DLMS model of the Virtual Distribution Equipment described above introduced abstract elements. This clause describes the structure of the data set object and the services which manage the data set.

The data set may be loaded or predefined. Services are provided to allow a DLMS client to handle the data set defined at the DLMS server.

7.1 Data set description

7.1.1 The data set object

The data set represents a set of the capabilities of the VDE, which is used for a specific purpose.

7.1.1.1 Attributes

The attributes of the data set are:

Object: Data Set
- key attribute: Data Set Name
- attribute: Scope Of Access (VDE-specific, VAA-specific)
- constraint Scope Of Access = VAA-specific
  - attribute: Executive VAA Name
- attribute: Scope May Change (TRUE, FALSE)
- attribute: Data Set Content
- attribute: Loadable (TRUE, FALSE)
- attribute: List of Task Invocation
- attribute: State (EMPTY, LOADING, READY)
- constraint: State=LOADING
  - attribute: Current Segment
  - attribute: Number of Segments
- constraint: State=READY
  - attribute: UpLoading (TRUE, FALSE)

The data set object is always predefined and its lifetime is always VDE. If the Executive VAA exists within the VDE, the scope of access of the data set object is always VAA-specific and the VAA Name attribute specifies the name of the Executive VAA.

7.1.1.2 Description

L'attribut Data Set Name (nom du jeu de données) identifie le jeu de données de manière unique dans le VDE.

L'attribut Scope of Access (champ d'accès) précise l'actuel champ d'accès du jeu de données. Ce champ d'accès fixe les règles d'accès à l'objet. Il doit être VDE-specific ou VAA-specific. Si l'attribut Scope Of Access contient la valeur VAA-specific, l'objet VAA associé est connu comme l'Executive VAA (VAA directeur). Son nom est spécifié dans l'attribut Executive VAA Name (nom du VAA directeur).

L'attribut Scope May Change (le domaine peut être modifié) indique si oui ou non le champ d'accès de l'objet considéré peut être changé à l'aide du service ChangeScope (modificateur de domaine) de DLMS.

L'attribut Data Set Content (contenu du jeu de données) décrit en langage naturel l'information contenue dans l'image chargée qui fait l'objet des services de chargement. La nature de cette information est particulière à l'implémentation. Pendant la phase de chargement, le contenu du jeu de données est constitué de plusieurs segments numérotés en ordre croissant (commençant à 1).

L'attribut Loadable (chargeable) définit si oui ou non un jeu de données peut être chargé dynamiquement.

L'attribut List of Task Invocation (liste d'invocations de tâches) identifie les TI qui utilisent actuellement le jeu de données. Plusieurs TI peuvent utiliser simultanément le même jeu de données, mais DLMS ne fournit pas de moyens de contrôle du partage des ressources ou des variables. Un partage équitable du jeu de données doit être fournit par les tâches elles même.

L'attribut State (état) décrit l'état du jeu de données. Le jeu de données peut prendre un des trois états suivants : EMPTY (vide), LOADING (chargement) ou READY (prêt). S'il n'est pas chargeable, le jeu de données est bloqué à l'état READY dès le début et pour toute la suite. Autrement, l'état initial du jeu de données peut être EMPTY ou READY. L'état EMPTY montre que le jeu de données doit être chargé avant d'être utilisé. L'état READY montre que le VDE-handler (gestionnaire VDE) peut utiliser le jeu de données. L'état LOADING est un état intermédiaire qui survient pendant le processus de chargement.

L'attribut Current Segment (segment actuel) identifie le numéro d'ordre du segment qui a été reçu et traité, dans la queue de chargement. Cet argument n'est défini que lorsque le jeu de données est dans l'état LOADING.

L'attribut Number of Segments (nombre de segments) rappelle le nombre total de segments attendus nécessaires au chargement du jeu de données. Cet attribut n'est défini que si le jeu de données est dans l'état LOADING.

L'attribut UpLoading (en cours de rapatriement) indique si un rapatriement est en cours ou non. Cet attribut est défini (et alors, un rapatriement peut arriver) seulement lorsque le jeu de données est dans l'état READY.

7.1.1.2 Description

The Data Set Name attribute identifies uniquely the data set within the VDE.

The Scope Of Access attribute specifies the current scope of access of the data set. This scope of access rules the object access. It shall be VDE-specific or VAA-specific. If the Scope Of Access attribute contains the VAA-specific value, the associated VAA object is known as the Executive VAA. Its name is specified in the Executive VAA Name attribute.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The Data Set Content attribute describes in natural language the information contained in the load image which is the subject of the load services. The nature of this information is implementation specific. During the loading phase, the data set content is composed of several segments numbered in increasing order (beginning with 1).

The Loadable attribute defines whether or not the data set may be dynamically loaded.

The List of Task Invocation attribute identifies the TI which are currently using the data set. Several TIs may use the data set at the same time but no means is provided by DLMS to control the sharing of resources or variables. Fair sharing of the data set must be provided within the tasks.

The State attribute describes the state of the data set. The data set may be in one of the three states: EMPTY, LOADING or READY. If it is not loadable, the data set is fixed in the READY state at the beginning and for every further moment. Otherwise, the data set initial state may be EMPTY or READY. The EMPTY state indicates that the data set must be loaded before being used. The READY state indicates that the VDE-handler may use the data set. The LOADING state is an intermediate state which occurs during the loading process.

The Current Segment attribute identifies the number of segments in the loading sequence which have been received and processed. This attribute is defined only when the data set is in the LOADING state.

The Number of Segments attribute recalls the total awaited number of segments necessary to perform the data set loading. This attribute is defined only when the data set is in the LOADING state.

The UpLoading attribute indicates whether or not an upload is being performed. This attribute is defined (and therefore, an Upload may occur) only when the data set is in the READY state.

7.1.1.3 Accès

SI l'Executive VAA existe dans le VDE, il contrôle l'accès à l'objet jeu de données. Le jeu de données est alors défini avec un champ d'accès spécifique VAA, et le VAA associé est connu comme étant l'Executive VAA. Dans ce cas, seul le client correspondant à cette Executive VAA, peut initialiser un service de chargement.

NOTE – L'Executive VAA contrôle aussi l'accès aux objets TI. S'il existe, c'est le seul à pouvoir utiliser avec succès les services de l'invocation de tâches et du gestionnaire de jeu de données dans l'environnement DLMS.

7.1.2 Contenu du jeu de données

Un objet défini dans le scope du VDE et dont la classe est une des suivantes peut être contenu par le jeu de données :

- variable nommée ;
- liste de variables nommées ;
- boîte à messages.

Le jeu de données contient aussi des informations telles que le code exécutable des tâches, qui n'est pas un objet DLMS.

7.1.3 Opérations sur le jeu de données

Les services InitiateLoad, LoadSegment, TerminateLoad, InitiateUpLoad, UpLoadSegment, TerminateUpLoad et GetDataAttribute agissent sur le jeu de données.

Les services de gestion du jeu de données permettent au client DLMS :

- de commencer une séquence de chargement pour charger un jeu de données dans le serveur DLMS ;
- d'exécuter et de vérifier une séquence de chargement ;
- de terminer une séquence de chargement ;
- de commencer une séquence de rapatriement pour récupérer un jeu de données du serveur DLMS ;
- d'exécuter et de vérifier une séquence de rapatriement ;
- de terminer une séquence de rapatriement ;
- d'obtenir les attributs du jeu de données.

7.1.1.3 Access

If the Executive VAA exists within the VDE, it controls the access to the data set object. The data set is then defined with a VAA-specific scope of access and the associated VAA is known to be the Executive VAA. In that case, only the corresponding DLMS client of the Executive VAA can initiate a load service.

NOTE – The Executive VAA also controls the access to the TI objects. If it exists, it is the only one that can successfully use the Task Invocation and the data set load management services in the DLMS environment.

7.1.2 Data set contents

An object defined within the scope of the VDE, and whose class is one of the following, may be contained in the data set:

- Named Variable;
- Named Variable List;
- Message Box.

The data set also contains information such as the executable code of the tasks, which is not a DLMS object.

7.1.3 Operations on the data set

The InitiateLoad, the LoadSegment, the TerminateLoad, the InitiateUpLoad, the UpLoadSegment, the TerminateUpLoad and the GetDataSetAttribute services operate on the data set.

The data set management services allow a DLMS client:

- to begin a load sequence to load the data set at the DLMS server;
- to perform and control a load sequence;
- to terminate a load sequence;
- to begin an upload sequence to obtain the data set from the DLMS server;
- to perform and control an upload sequence;
- to terminate an upload sequence;
- to obtain the attributes of the data set.

La séquence de chargement d'un jeu de données sert à transférer le contenu du jeu de données du client DLMS au serveur DLMS. Le client DLMS initialise cette séquence en émettant une demande de service InitiateLoad. Puis le client DLMS émet, en fonction des besoins, une ou plusieurs demandes de service LoadSegment, et finalement le serveur DLMS confirme l'avancement du processus.

La séquence de rapatriement d'un jeu de données sert à transférer le contenu du jeu de données du serveur DLMS au client DLMS. Le client DLMS initialise cette séquence en émettant une demande de service InitiateUpLoad. Puis le client DLMS émet, en fonction des besoins, une ou plusieurs demandes de service UpLoadSegment, et finalement le serveur DLMS confirme l'avancement du processus.

7.2 Service InitiateLoad (initialisation du chargement)

7.2.1 Objet

Le service InitiateLoad est sollicité par le client DLMS pour ordonner au serveur DLMS de charger le jeu de données et éventuellement de le renommer.

7.2.2 Structure

Tableau 7 - Le service InitiateLoad

Service InitiateLoad	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Number of Segments	M	M(=)		
Data Set Content	U	U(=)		
Result(+)			S	S(=)
Already Exists			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.2.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service InitiateLoad.

Le paramètre Data Set Name (nom du jeu de données), de type ObjectName (nom d'objet), indique le nouveau nom du jeu de données à charger.

Le paramètre Number of Segments (nombre de segments), de type Unsigned16, spécifie la longueur totale, exprimée en nombre de segments, des données qui doivent être chargées. Ce nombre est utilisé par le serveur DLMS pour connaître la fin d'un processus de chargement. La combinaison du paramètre Number of Segments et la longueur maximale de chaque segment fournie par la spécification d'accompagnement fournissent une valeur approximative de la taille du jeu de données tout entier.

Le paramètre Data Set Content (contenu du jeu de données), de type Visible String (chaîne visible), décrit en langage naturel le contenu du jeu de données.

The data set load sequence is used to transfer a data set content from the DLMS client to the DLMS server. The DLMS client initiates this sequence by issuing an InitiateLoad service request. Then, the DLMS client issues one or more LoadSegment service requests (as required); finally, the DLMS server confirms the advance of the process.

The data set upload sequence is used to transfer a data set content from the DLMS server to the DLMS client. The DLMS client initiates this sequence by issuing an InitiateUpLoad service request. Then, the DLMS client issues one or more UpLoadSegment service requests (as required); finally, the DLMS server confirms the advance of the process.

7.2 InitiateLoad service

7.2.1 Purpose

The InitiateLoad service is requested by the DLMS client to instruct the DLMS server to load the data set and eventually to rename it.

7.2.2 Structure

Table 7 - The InitiateLoad service

InitiateLoad service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Number of Segments	M	M(=)		
Data Set Content	U	U(=)		
Result(+)			S	S(=)
Already Exists			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.2.3 Parameters

The Argument parameter conveys the specific parameters of the InitiateLoad service request.

The Data Set Name parameter, of type ObjectName, indicates the new name of the data set which is to be loaded.

The Number of Segments parameter, of type Unsigned16, specifies the total length, expressed in number of segments, of the data that are to be loaded. This number is used by the DLMS server to know the end of the loading process. The combination of the Number of Segments parameter and the maximum length of each segment specified in the companion specification provides an approximate value of the total data set size.

The Data set Content parameter, of type Visible String, describes in natural language the content of the data set.

Le paramètre Result(+) indique que la demande InitiateLoad a abouti.

Le paramètre Already Exists (existe déjà), de type BOOLEAN (booléen), spécifie si oui (TRUE) ou non (FALSE) le nouveau nom du jeu de données est identique à celui qui existe déjà.

Le paramètre Result(-) indique que la demande de service émise auparavant n'a pas abouti. Le paramètre Error Type (type d'erreur) fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.2.4 Procédures de service

Quand il reçoit une primitive indication de Initiateload, le serveur DLMS vérifie les attributs du jeu de données.

Le service échoue si on rencontre l'un des cas suivants :

- l'attribut Loadable (chargeable) du jeu de données n'est pas à TRUE (vrai) ;

- les restrictions d'accès ne sont pas satisfaites ;

- l'attribut State (état) est à LOADING (chargement) ;

- l'espace réservé n'est pas suffisant quand on considère le nombre de segments attendus.

Une réponse erreur spécifiant le type d'erreur approprié est renvoyée.

NOTE – Les restrictions d'accès sont héritées du champ d'accès du jeu de données. Les restrictions d'accès ne sont pas satisfaites si le champ d'accès est VAA-specific et si l'Executive VAA associée n'est pas celle qui a demandé le service.

Si le service réussit, l'état du jeu de données est positionné à LOADING, tous les objets existants dans le jeu de données sont effacés et l'état des TI qui utilisent actuellement le jeu de données passe à UNUSABLE (inutilisable). Le champ d'accès de ces TI et du jeu de données devient VAA-specific. l'Executive VAA associée est la VAA du client demandeur.

The Result(+) parameter indicates that the requested InitiateLoad service has succeeded.

The Already Exists parameter, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) the new data set name is the same as the one that already exists.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.2.4 Service procedure

When receiving an InitiateLoad indication primitive, the DLMS server verifies the attributes of the data set.

The service fails if one of the following statements is met:

- the Loadable attribute of the corresponding data set is not set to TRUE;

- the access restrictions are not satisfied;

- the State attribute is set to LOADING;

- the reserved size is not large enough with regard to the number of awaited segments.

An error response specifying the appropriate error type is then returned.

NOTE – The access restrictions are inherited from the scope of access of the data set. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

If the service succeeds, the state of the data set is set to LOADING, all existing objects in the data set are deleted and the state of the TIs that currently use the data set is set to UNUSABLE. The scope of access of these TIs and of the data set is set to VAA-specific. The associated Executive VAA is the VAA of the requesting DLMS client.

7.3 Le service LoadSegment *(chargement d'un segment)*

7.3.1 Objet

Ce service est sollicité par un client DLMS pour charger un segment d'information dans le serveur DLMS.

7.3.2 Structure

Tableau 8 - Le service LoadSegment

Service LoadSegment	Req	Ind	Resp	Conf
Argument	M	M(=)		
Segment Number	M	M(=)		
Segment	M	M(=)		
Result(+)			S	S(=)
Current Segment			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.3.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques à la demande de service LoadSegment.

Le paramètre Segment Number (numéro de segment), de type Unsigned16, indique le numéro du segment (commençant à 1 pour le premier segment) contenu actuellement dans la primitive demande. Il peut être utilisé pour récupérer les erreurs de communication.

Le paramètre Segment, de type OCTET STRING (chaîne d'octets), contient les données à charger. Tous les segments attachés par ordre de numéro de segment croissant constituent le jeu de données.

NOTE – Il n'est fait aucune supposition dans cette norme concernant la nature de l'information contenue dans les segments.

Le paramètre Result(+) indique que la demande de service a abouti.

7.3 LoadSegment service

7.3.1 Purpose

This service is requested by the DLMS client to load a segment of information at the DLMS server.

7.3.2 Structure

Table 8 - The LoadSegment service

LoadSegment service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Segment Number	M	M(=)		
Segment	M	M(=)		
Result(+)			S	S(=)
Current Segment			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.3.3 Parameters

The Argument parameter conveys the specific parameters of the LoadSegment Service request.

The Segment Number parameter, of type Unsigned16, indicates the number of the segment (beginning with 1 for the first segment) currently contained in the request primitive. It can be used to recover from communication failures.

The Segment parameter, of type OCTET STRING, contains the data that are to be loaded. All the segments appended in increasing order of their segment number constitute the data set.

NOTE – No assumption is made in this standard about the nature of the information in the segments.

The Result(+) parameter indicates that the requested service has succeeded.

Le paramètre Current Segment (segment actuel), de type Unsigned16, rappelle le numéro du segment reçu.

Le paramètre Result(-) indique que le service demandé précédemment a échoué. Le paramètre Error Type (type d'erreur) indique la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.3.4 Procédure de service

Tant qu'un processus de chargement de données n'est pas terminé, le client DLMS demandeur affecte aux paramètres les valeurs demandées par la confirmation LoadSegment reçue précédemment, s'il y en a une.

A la réception d'une indication d'un LoadSegment, le serveur DLMS compare le paramètre Segment Number avec l'attribut Current Segment du jeu de données. Si ce n'est pas le segment attendu, une réponse négative est renvoyée et le paramètre Current Segment reçoit le numéro du dernier segment traité avec succès.

Si c'est le segment attendu, le serveur DLMS interprète les bits reçus dans le paramètre Segment conformément au format du chargement du jeu de données ; il les stocke alors comme approprié. A la suite d'un traitement réussi des paramètres de LoadSegment, le serveur DLMS met à jour l'attribut Current Segment du jeu de données et renvoie une réponse de succès avec le paramètre Current Segment positionné sur le numéro du dernier segment traité avec succès. Si le processus de chargement a échoué, une réponse négative est renvoyée.

7.4 Service TerminateLoad *(terminer le chargement)*

7.4.1 Objet

Le service TerminateLoad est utilisé par le client DLMS pour informer le serveur DLMS que la séquence de chargement est terminée. Ce service est offert par DLMS pour réaliser les tables d'état de la gestion du chargement.

The Current Segment parameter, of type Unsigned16, recalls the number of the received segment.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.3.4 Service procedure

While the data load process is not over, the requesting DLMS client sets the parameter values as required in the previously received LoadSegment confirm, if it exists.

On receipt of a LoadSegment indication, the DLMS server compares the Segment Number parameter with the Current Segment attribute of the data set. If it is not the awaited segment, a negative response is returned with the Current Segment parameter set to the last successfully processed segment number.

If it is the awaited segment, the DLMS server interprets the bits received in the Segment parameter according to the Data Set Load format; it then stores them as appropriate. Following successful processing of the LoadSegment parameters, the DLMS server updates the Current Segment attribute of the data set and returns a success response with the Current Segment parameter set to the last successfully processed segment. If the loading process has failed, a negative response is returned.

7.4 TerminateLoad service

7.4.1 Purpose

The TerminateLoad service is used by the DLMS client to inform the DLMS server that the load sequence is over. This service is offered by DLMS to complete the load management state tables.

7.4.2 Structure

Tableau 9 - Le service TerminateLoad

Service TerminateLoad	Req	Ind	Resp	Conf
Argument	M	M(=)		
Result(+)			S	S(=)
New State			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.4.3 Paramètres

Le paramètre Argument convoie une valeur NULL car il n'y a pas de paramètres transmis au service TerminateLoad.

Le paramètre Result(+) indique que la demande a abouti.

Le paramètre New State (nouvel état), de type ENUMARATED (énumération), spécifie le prochain état du serveur DLMS après avoir complété les services de gestion du chargement.

Le paramètre Result(-) indique que le service précédemment demandé a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.4.4 Procédure de service

A la réception d'une primitive indication du service TerminateLoad, le jeu de données sort de l'état LOADING. Si les segments ont été chargés correctement, le prochain état est READY. Si les segments n'ont pas été chargés correctement, l'état suivant est EMPTY. Dans tous les cas, le serveur DLMS renvoie une réponse positive en spécifiant le prochain état approprié. Si le serveur DLMS n'était pas dans l'état LOADING, il renvoie une réponse négative.

7.5 Service InitiateUpLoad (initialisation du rapatriement)

7.5.1 Objet

Le service InitiateUpLoad est sollicité par le client DLMS pour ordonner le serveur DLMS de rapatrier le jeu de données, c'est-à-dire de transférer son contenu (en totalité ou en partie) vers le client DLMS.

7.4.2 Structure

Table 9 - The TerminateLoad service

TerminateLoad service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Result(+)			S	S(=)
New State			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.4.3 Parameters

The Argument parameter conveys a NULL value, because no parameter is transmitted with the TerminateLoad service.

The Result(+) parameter indicates that the requested service has succeeded.

The New State parameter, of type ENUMERATED, specifies the next state of the DLMS server after completing the Load management services.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.4.4 Service procedure

When receiving a TerminateLoad indication service primitive, the data set transits out of the LOADING state. If the segments were correctly loaded, the next state is READY. If the segments were not correctly loaded, the next state is EMPTY. In both cases the DLMS server returns a positive response specifying the proper next state. If the DLMS server was not currently in the LOADING state, a negative response is returned.

7.5 InitiateUpLoad service

7.5.1 Purpose

The InitiateUpLoad service is requested by the DLMS client to instruct the DLMS server to upload the data set, that is to transfer its contents (in whole or in part) to the DLMS client.

7.5.2 Structure

Tableau 10 - Le service InitiateUpLoad

Service InitiateUpLoad	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Result(+)			S	S(=)
Number of Segments			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.5.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service InitiateUpLoad.

Le paramètre Data Set Name (nom du jeu de données), de type ObjectName (nom d'objet), indique le nouveau nom du jeu de données à rapatrier.

Le paramètre Result(+) indique que la demande InitiateUpLoad a abouti.

Le paramètre Number of Segments (nombre de segments), de type Unsigned16, spécifie la longueur totale, exprimée en nombre de segments, des données qui doivent être rapatriées. Ce nombre est utilisé par le serveur DLMS pour connaître la fin d'un processus de rapatriement. La combinaison du paramètre Number of Segments et de la longueur maximale de chaque segment fournie par la spécification d'accompagnement fournit une valeur approximative de la taille du jeu de données tout entier.

Le paramètre Result(-) indique que la demande de service émise auparavant n'a pas abouti. Le paramètre Error Type (type d'erreur) fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.5.4 Procédures de service

Quand il reçoit une primitive indication de InitiateUpLoad, le serveur DLMS vérifie les attributs du jeu de données, puis calcule le nombre de segments requis afin de transmettre le contenu du jeu de données demandé par le client DLMS.

7.5.2 Structure

Table 10 - The InitiateUpLoad service

InitiateUpLoad service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Result(+)			S	S(=)
Number of Segments			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.5.3 Parameters

The Argument parameter conveys the specific parameters of the InitiateUpLoad Service request.

The Data Set Name parameter, of type ObjectName, indicates the name of the existing data set which is to be uploaded.

The Result(+) parameter indicates that the requested InitiateUpLoad service has succeeded.

The Number of Segments parameter, of type Unsigned16, specifies the total length, expressed in number of segments, of the data that are to be uploaded. This number is used by DLMS server to know the end of an uploading. The combination of the Number of Segments parameter and the maximum length of each segment specified in the companion specification provides an approximate value of the total data set size.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.5.4 Service procedure

When receiving an InitiateUpLoad indication primitive, the DLMS server verifies the attributes of the data set, then computes the number of segments required in order to transmit the contents of the data set required by the DLMS client.

Le service échoue si on rencontre l'un des cas suivants :

- le paramètre Data Set Name n'est pas égal à l'attribut Data Set Name de l'objet jeu de données défini dans le serveur;

- l'attribut UpLoadable (chargeable) n'est pas à TRUE (vrai) ;

- les restrictions d'accès ne sont pas satisfaites ;

- l'attribut State (état) n'est pas à READY ;

- l'espace réservé n'est pas suffisant quand on considère le nombre de segments attendus.

On renvoie une réponse erreur spécifiant le type d'erreur approprié.

NOTE – Les restrictions d'accès sont héritées du champ d'accès du jeu de données. Les restrictions d'accès ne sont pas satisfaites si le champ d'accès est VAA-specific et si l'Executive VAA n'est pas celle qui a demandé le service.

Attention :

Les interactions d'un chargement et d'un rapatriement d'un jeu de données peuvent conduire à des incohérences dans les informations rapatriées, à moins de surveiller ceci soigneusement.

7.6 Le service UpLoadSegment *(rapatriement d'un segment)*

7.6.1 Objet

Ce service est sollicité par un client DLMS pour rapatrier un segment d'information du serveur DLMS.

The service fails if one of the following statements is met:

- the Data Set Name parameter is not equal to the Data Set Name attribute of the data set object defined in the server;

- the UpLoadable attribute is not set to TRUE;

- the access restrictions are not satisfied;

- the State attribute is not set to READY;

- the reserved space is insufficient if considering the number of awaited segments.

An error response specifying the appropriate error type is then returned.

NOTE – The access restrictions are inherited from the scope of access of the data set. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

Warning:

Interleaving of data set load and upload services may lead to incoherences in the uploaded information, unless carefully controlled.

7.6 UpLoadSegment service

7.6.1 Purpose

This service is requested by the DLMS client to upload a segment of information from the DLMS server.

7.6.2 Structure

Tableau 11 - Le service UpLoadSegment

Service UpLoadSegment	Req	Ind	Resp	Conf
Argument	M	M(=)		
Segment Number	M	M(=)		
Result(+)			S	S(=)
Current Segment			M	M(=)
Segment			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.6.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques à la demande de service UpLoadSegment.

Le paramètre Segment Number (numéro de segment), de type Unsigned16, indique le numéro du segment (la numérotation commençant à 1) actuellement requis. Il peut être utilisé pour récupérer les erreurs de communication.

Le paramètre Result(+) indique que la demande de service a abouti.

Le paramètre Current Segment (segment actuel), de type Unsigned16, rappelle le numéro du segment en cours de rapatriement.

Le paramètre Segment, de type OCTET STRING (chaîne d'octets), contient les données à rapatrier. Tous les segments attachés dans l'ordre croissant de leur numéros de segments constituent la partie du jeu de données devant être rapatriée.

NOTE – Il n'est fait aucune supposition dans cette norme concernant la nature de l'information.

Le paramètre Result(-) indique que le service demandé précédemment a échoué. Le paramètre Error Type (type d'erreur) indique la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.6.2 Structure

Table 11 - The UpLoadSegment service

UpLoadSegment service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Segment Number	M	M(=)		
Result(+)			S	S(=)
Current Segment			M	M(=)
Segment			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.6.3 Parameters

The Argument parameter conveys the specific parameters of the UpLoadSegment Service request.

The Segment Number parameter, of type Unsigned16, indicates the number of the segment (the numbering of segments begins with 1) currently being requested. It can be used to recover from communication failures.

The Result(+) parameter indicates that the requested service has succeeded.

The Current Segment parameter, of type Unsigned16, recalls the number of the segment being uploaded.

The Segment parameter, of type OCTET STRING, contains the data that are to be uploaded. All the segments appended in increasing order of their segment number constitute the part of the data set which is to be uploaded.

NOTE – No assumption is made in this standard about the nature of the information in the segments.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.6.4 Procédure de service

Tant qu'un processus de rapatriement de données n'est pas terminé, le client DLMS demandeur affecte aux paramètres les valeurs demandées par la confirmation UpLoadSegment reçue précédemment, s'il y en a une.

A la réception d'une indication d'un UpLoadSegment, le serveur DLMS envoie le segment demandé, si possible. Si une erreur arrive, une réponse négative est retournée.

7.7 Service TerminateUpLoad *(terminer le rapatriement)*

7.7.1 Objet

Le service TerminateUpLoad est utilisé par le client DLMS pour informer le serveur DLMS que la séquence de chargement est terminée. Ce service est offert par DLMS pour réaliser les tables d'état de la gestion du rapatriement.

7.7.2 Structure

Tableau 12 - Le service TerminateUpLoad

Service TerminateUpLoad	Req	Ind	Resp	Conf
Argument	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.7.3 Paramètres

Le paramètre Argument convoie une valeur NULL car il n'y a pas de paramètres transmis au service TerminateUpLoad.

Le paramètre Result(+) indique que la demande a abouti.

Le paramètre Result(-) indique que le service précédemment demandé a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

7.7.4 Procédure de service

A la réception d'une primitive indication du service TerminateUpLoad, le jeu de données sort de l'état UPLOADING. Si le segment a été chargé correctement, le prochain état est READY. Si le serveur DLMS n'était pas dans l'état UPLOADING, il renvoie une réponse négative.

7.6.4 Service procedure

While the data upload process is not over, the requesting DLMS client sets the parameter values as required in the previously received UpLoadSegment confirm, if it exists.

On receipt of a UpLoadSegment indication, the DLMS server sends the segment requested, if possible. If any error occurs, a negative response is returned.

7.7 TerminateUpLoad service

7.7.1 Purpose

The TerminateUpLoad service is used by the DLMS client to inform the DLMS server that the upload sequence is over. This service is offered by DLMS to complete the upload management state tables.

7.7.2 Structure

Table 12 - The TerminateUpLoad service

TerminateUpLoad service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.7.3 Parameters

The Argument parameter conveys a NULL value because no parameter is transmitted with the TerminateUpLoad service.

The Result(+) parameter indicates that the requested service has succeeded.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.7.4 Service procedure

When receiving a TerminateUpLoad indication service primitive, the data set transits out of the UPLOADING state. If the segment has been correctly loaded, the next state is READY. If the DLMS server was not currently in the UPLOADING state, a negative response is returned.

7.8 Le service GetDataSetAttribute (obtenir les attributs du jeu de données)

7.8.1 Objet

Le service GetDataSetAttribute sert à demander au serveur DLMS de renvoyer tous les attributs du jeu de données.

7.8.2 Structure

Tableau 13 - Le service GetDataSetAttribute

Service GetDataSetAttribute	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Result(+)			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Data Set Content			M	M(=)
Loadable			M	M(=)
List of Task Invocation			M	M(=)
State			M	M(=)
Current Segment			C	C(=)
Number of Segments			C	C(=)
Uploading			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.8.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques des demandes de service GetDataSetAttribute.

Le paramètre DataSetName (non du jeu de données), de type ObjectName (nom d'objet), véhicule le nom du jeu de données dont on demande les attributs.

Le paramètre Result(+) indique que la demande de service a été satisfaite.

Le paramètre Scope of Access (champ d'accès), de type ScopeOfAccess, spécifie le champ d'accès de l'objet. Il doit être VDE-specific ou VAA-specific. Si ce paramètre contient la valeur VAA-specific, l'objet VAA associé est connu comme Executive VAA.

7.8 GetDataSetAttribute service

7.8.1 Purpose

The GetDataSetAttribute service is used to request that a DLMS server returns all the attributes associated with the data set.

7.8.2 Structure

Table 13 - The GetDataSetAttribute service

GetDataSetAttribute service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Data Set Name	M	M(=)		
Result(+)			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Data Set Content			M	M(=)
Loadable			M	M(=)
List of Task Invocation			M	M(=)
State			M	M(=)
Current Segment			C	C(=)
Number of Segments			C	C(=)
Uploading			C	C(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.8.3 Parameters

The Argument parameter conveys the specific parameters of the GetDataSetAttribute Service request.

The Data Set Name parameter, of type ObjectName, conveys the data set name of which attributes are requested.

The Result(+) parameter indicates that the requested service has succeeded.

The Scope of Access parameter, of type ScopeOfAccess, specifies the scope of access of the related object. It must be VDE-specific or VAA-specific. If the Scope Of Access parameter contains the VAA-specific value, the associated VAA object name is known as the Executive VAA.

Le paramètre Scope May Change (le champ peut changer), de type BOOLEAN (booléen), indique si oui (TRUE) ou non (FALSE) le champ d'accès de l'objet considéré peut être modifié à l'aide du service ChangeScope de DLMS.

Le paramètre DataSetContent (contenu du jeu de données), de type VisibleString (chaîne visible), décrit en langage naturel le contenu du jeu de données.

Le paramètre Loadable (Chargeable), de type BOOLEAN (booléen), définit si oui (TRUE) ou non (FALSE) l'objet jeu de données peut être chargé en utilisant les services de chargement de DLMS.

Le paramètre List of Task Invocation (liste d'invocations de tâches), de type SEQUENCE OF Object Name (séquence de nom d'objet), identifie les TI qui emploient actuellement le jeu de données.

Le paramètre State (état), de type CHOICE (choix), décrit l'état du jeu de données. Chaque jeu de données peut être dans l'un de ces trois états : EMPTY (vide), LOADING (chargement), READY (prêt).

Le paramètre Current Segment (segment en cours), de type Unsigned16, identifie le numéro des segments dans la séquence de chargement qui ont été reçus et traités. Cet attribut n'est défini que lorsque le jeu de données est dans l'état LOADING (chargement).

Le paramètre Number of Segments (nombre de segments), de type Unsigned16, rappelle le nombre de segments nécessaires au chargement du jeu de données en cours de chargement. Cet attribut n'est défini que lorsque le jeu de données est dans l'état LOADING (chargement).

L'attribut UpLoading (en cours de rapatriement) de type BOOLEAN (booléen), indique qu'un rapatriement est en cours. Cet attribut est défini (et alors, un rapatriement peut arriver) seulement lorsque le jeu de données est dans l'état READY.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type (type d'erreur) fournit la raison de l'erreur. Le paramètre Error Type est décrit en détail en A.5.

7.8.4 Procédure de service

La demande identifie le nom du jeu de données. Le serveur DLMS vérifie que le nom du jeu de données est valide. Si le domaine d'accès du jeu de données est spécifique VAA, il vérifie aussi que la VAA pour qui le service est demandé est l'Executive VAA. Si ces vérifications sont satisfaites, il renvoie les attributs définis au début de ce chapitre. Autrement, il renvoie une réponse négative.

The Scope May Change parameter, of type BOOLEAN, defines whether (TRUE) or not (FALSE) the scope of access of the related object may change using the DLMS ChangeScope service.

The Data Set Content parameter, of type VisibleString, describes in natural language the content of the data set.

The Loadable parameter, of type BOOLEAN, defines whether (TRUE) or not (FALSE) the data set object may be loaded using the DLMS load services.

The List of Task Invocation parameter, of type SEQUENCE OF ObjectName, identifies the TIs which currently use the data set.

The State parameter, of type CHOICE, describes the state of the data set. Each Data Set may be in one of the three states: EMPTY, LOADING or READY.

The Current Segment parameter, of type Unsigned16, identifies the number of segments in the loading sequence which have been received and processed. This attribute is defined only when the data set is in the LOADING state.

The Number of Segments parameter, of type Unsigned16, recalls the awaited number of segments necessary to perform the data set loading. This attribute is defined only when the data set is in the LOADING state.

The UpLoading attribute, of type BOOLEAN, indicates that an upload is being performed. This attribute is defined (and therefore, an Upload may occur) only when the data set is in the READY state.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

7.8.4 Service procedure

The request identifies the data set name. The DLMS server verifies that the data set name is valid; if the Scope of Access of the data set is VAA-specific, it also checks that the VAA on which the service is requested is the VAA-Executive. If these checks are successful, it returns the attributes defined at the beginning of this chapter. Otherwise, a negative response is returned.

8 Service de gestion de VAA

8.1 Description de VAA

8.1.1 Objet

L'objet VAA est spécial dans l'environnement DLMS. Une VAA (association d'applications) est à la fois un ensemble d'objets partageant le même domaine d'accès et un objet qui identifie de manière unique le client DLMS. L'objet VAA fournit les moyens de restreindre l'accès à beaucoup d'objets définis dans le VDE au client DLMS "autorisé". Cela assure la sécurité et la confidentialité des données contenues dans le VDE. Accès signifie obtenir ou modifier les attributs ou la valeur des objets, sauf leur nom qui est toujours accessible à tout le monde à l'aide d'un service GetNameList.

Le contenu de l'objet VAA est l'ensemble des noms de tous les objets définis dans le VDE et dont le champ est VAA-specific.

Comme on le voit en 5.2, un objet VAA est créé chaque fois qu'une primitive réponse Initiate a été reçue avec succès. L'objet VAA est lié directement à l'existence du contexte VAA correspondant. L'objet VAA identifie le client DLMS dans le VDE.

L'accès réservé à un objet est fourni par l'utilisation du champ d'accès défini pour chaque objet DLMS. S'il est VAA-specific, le paramètre VAA Name contient le nom de l'objet VAA qui contrôle l'accès à cet objet. Cet objet VAA représente lui-même le client DLMS dans le VDE. Quand un accès à un objet DLMS est demandé à l'aide des services de DLMS, le domaine d'accès est vérifié. S'il est VAA-specific, on compare le VAA Name spécifié dans le paramètre Name avec le nom du VAA qui représente le demandeur DLMS. Il faut que l'égalité soit prouvée avant d'autoriser l'accès.

Sur ces mêmes bases, un objet VAA peut contrôler les fonctions de chargement du VDE-handler et les TI. Un tel VAA est dit Executive VAA. Dans un VDE il n'y a pas plus d'un seul Executive VAA.

8 VAA management service

8.1 VAA description

8.1.1 Purpose

The VAA object is special in the DLMS environment. A VAA is both a set of objects that share a common scope of access and an object that uniquely identifies the DLMS client. The VAA object provides the means to restrict the access to many objects defined at the VDE to the "allowed" DLMS client. This provides security and confidentiality of the data contained in the VDE. Access means getting or changing the attributes or the value of an object, with the exception of its name which can always be obtained by anyone through the GetNameList service.

The content of the VAA object is the set of the names of all the objects defined within the VDE that have this VAA-specific scope.

As shown in 5.2, a VAA object is created every time that a successful Initiate response primitive is received. The VAA object is directly bound to the existence of a corresponding VAA context. The VAA object then identifies the DLMS client at the VDE.

Reserved access to an object is provided through the use of the scope of access defined for each DLMS object. If VAA-specific, the VAA Name parameter contains the name of the VAA object that controls the access to the object. This VAA object itself represents the DLMS client within the VDE. When an access to a DLMS object is requested using DLMS services, the scope of access is checked. When VAA-specific, the VAA Name specified in the Name parameter and the name of the VAA that represents the DLMS requester are compared. Equality must be achieved before authorizing the access.

On the same basis, a VAA object may control the Load functions of the VDE-handler and the TIs. Such a VAA is known as the Executive VAA. In a VDE, there is at most one Executive VAA.

8.1.2 L'objet VAA

8.1.2.1 Attributs

Les attributs de VAA sont :

Object : VAA
key attribute : VAA Name
attribute : Scope Of Access (VDE-specific)
attribute : Executive (TRUE, FALSE)
attribute : Abortable (TRUE, FALSE)
attribute : List of Named Variable
attribute : List of Named Variable List
attribute : List of Message Box

8.1.2.2 Description

L'attribut VAA Name identifie de manière unique dans un VDE l'objet VAA.

Le champ d'accès de l'objet VAA est toujours VDE-specific.

L'attribut Executive (directeur) spécifie si le VAA contrôle ou non les fonctions de chargement du VDE-handler et les différentes TI définies dans le VDE. Dans un VDE, un seul VAA à la fois, peut avoir l'attribut Executive à TRUE (vrai). Tous les autres VAA ont leur attribut Executive à FALSE (faux).

L'attribut Abortable spécifie si le service Abort peut être utilisé pour terminer la VAA.

NOTE – Puisque le service Abort peut quelquefois (par exemple en cas de défaillances internes de l'équipement) être l'unique moyen de récupérer le contrôle d'un VDE, il convient de n'utiliser le positionnement de l'attribut Abortable à FALSE qu'avec les plus grandes précautions.

L'attribut List of Named Variable (liste de variables désignées) contient la liste des objets variables désignées contenus dans la VAA. Ce sont toutes les variables désignées dans le VDE avec un champ VAA-specific.

L'attribut List of Named Variable List contient la liste des objets liste des variables désignées contenus dans la VAA. Ce sont toutes les listes de variables désignées dans le VDE avec un champ VAA-specific.

L'attribut List of Message Box (liste des boîtes à messages) contient la liste des objets boîtes à messages contenus dans le VDE. Ce sont toutes les boîtes à messages définies dans le VDE avec un champ VAA-specific.

8.1.2 The VAA object

8.1.2.1 Attributes

The attributes of the VAA are:

Object: VAA
key attribute: VAA Name
attribute: Scope Of Access (VDE-specific)
attribute: Executive (TRUE, FALSE)
attribute: Abortable (TRUE, FALSE)
attribute: List of Named Variable
attribute: List of Named Variable List
attribute: List of Message Box

8.1.2.2 Description

The VAA Name attribute identifies uniquely within a VDE the VAA object.

The scope of access of a VAA object is always VDE-specific.

The Executive attribute specifies whether or not the VAA controls the load functions of the VDE-handler and the different TIs that are defined within the VDE. In the VDE, only one VAA at a time may have its Executive attribute set to TRUE. All the other VAAs have their Executive attribute set to FALSE.

The Abortable attribute indicates whether the Abort service may be used to terminate the VAA.

NOTE – Due to the fact that the Abort service may at times (for example internal failures of the equipment) be the only means left to regain control of the VDE, setting the Abortable attribute to FALSE should be performed with the utmost caution.

The List of Named Variable attribute contains the list of the Named Variable objects contained in the VAA. These are all the named variables defined within the VDE with this VAA-specific scope.

The List of Named Variable List attribute contains the list of the Named Variable List objects contained in the VAA. These are all the lists of named variables defined within the VDE with this VAA-specific scope.

The List of Message Box attribute contains the list of the Message Box objects contained in the VAA. These are all the message boxes defined within the VDE with this VAA-specific scope.

8.1.3 Opérations sur les VAA

Le service VAA Management (gestion de la VAA) fait partie du service ChangeScope. Ce service permet au client DLMS de changer les champs d'accès aux objets de DLMS, moyennant le respect de certaines règles de protection.

Pour obtenir des informations sur des variables appartenant à VAA, on peut utiliser le service GetNameList, tel que décrit dans l'article 6.

8.2 Le service ChangeScope *(modifier champ)*

8.2.1 Objet

Le service ChangeScope est utilisé pour modifier les champ d'accès des objets spécifiés dans la primitive de service. Parce que l'objet VAA et le champ d'accès représentent le contrôle qu'un client DLMS applique aux objets de VDE, le service ChangeScope permet au client DLMS de prendre, d'abandonner ou de transmettre ce contrôle sur les objets de VDE. Un utilisateur peut modifier le champ de ses objets ou de ceux de VDE. Cependant, un utilisateur ne peut pas changer le champ d'objets qui ne sont pas dans son propre VAA. Des règles plus contraignantes peuvent être imposées par le VDE-handler.

8.2.2 Structure

Tableau 14 - Le service ChangeScope

Service ChangeScope	Req	Ind	Resp	Conf
Argument	M	M(=)		
New Scope	M	M(=)		
New VAA	U	U(=)		
Changed Scope Objects	U	U(=)		
Result(+)			S	S(=)
List of ChangeScope Results			M	M(=)
Success			S	S(=)
ChangeScope Failure			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

8.2.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service ChangeScope.

8.1.3 Operations on the VAAs

The VAA Management service is the ChangeScope service. This service allows the DLMS client to change the scope of access of any DLMS object, observing some protection rules.

To obtain information on variables owned by a VAA, the GetNameList service described in clause 6 may be used.

8.2 ChangeScope service

8.2.1 Purpose

The ChangeScope service is used to change the scope of access of the objects specified in the service primitive. Because the VAA object and the scope of access represent the control of a DLMS client on VDE objects, the ChangeScope service allows the DLMS client to take control, to release control, or to transmit control on VDE objects. A user may change the scope of its objects or those of the VDE. However, the user may not change the scope of objects that are within another VAA than its own. More constraining rules may be enforced by the VDE-handler.

8.2.2 Structure

Table 14 - The ChangeScope service

ChangeScope service	Req	Ind	Resp	Conf
Argument	M	M(=)		
New Scope	M	M(=)		
New VAA	U	U(=)		
Changed Scope Objects	U	U(=)		
Result(+)			S	S(=)
List of ChangeScope Results			M	M(=)
Success			S	S(=)
ChangeScope Failure			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

8.2.3 Parameters

The Argument parameter conveys the specific parameters of the ChangeScope service request.

Le paramètre New Scope (nouveau champ), du type BOOLEAN (booléen) indique le nouveau champ d'accès souhaité pour les objets spécifiés dans la demande. Comme défini ci-dessus, un champ d'accès doit être VDE-specific ou VAA-specific. Si le champ d'accès est VAA-specific, le VAA Name du destinataire doit être aussi fourni (dans le paramètre New VAA) quand ce n'est pas celui de l'utilisateur demandeur; autrement, ce n'est pas obligatoire.

Le paramètre optionnel Change Scope Objects (changer le domaine des objets), s'il est présent, spécifie une liste d'objets dont il faut changer le champ. Si ce paramètre contient le nom d'un jeu de données ou celui de une ou plusieurs invocations de tâches, ou s'il est absent, il est alors nécessaire de changer l'Executive VAA. Le paramètre Result(+) indique que la demande a abouti.

Le paramètre List of Change Scope Results (liste des résultats de changement de champ) contient les résultats d'une opération de changement de champ pour chacun des objets, dans l'ordre spécifié par le paramètre Change Scope Objects de la primitive demande. Chaque élément signifie soit un succès du changement de champ pour cet objet (dans le paramètre Success), soit une raison de l'échec (dans le paramètre Change Scope Failure). Si le paramètre Change Scope Objects n'était pas présent dans la demande (ce qui veut dire que seul un changement dans l'Executive VAA est demandé), le paramètre List of Change Scope Results doit contenir précisément un seul paramètre qui donne le résultat du changement de l'Executive VAA.

Le paramètre Result(-), du type Service Error, indique que le service demandé auparavant a échoué.

8.2.4 Procédure de service

A la réception d'une primitive request de ChangeScope, le client DLMS est identifié avec le VAA Name correspondant. Pour chaque nom soumis dans le paramètre Change Scope Objects, le serveur DLMS vérifie la validité du changement de l'accès et tente de changer le champ de l'objet. Si le paramètre Change Scope Objects est absent ou s'il contient le nom du jeu de données ou les noms d'invocation de tâches, le serveur DLMS vérifie la validité du changement de l'Executive VAA. Si cette vérification est positive, il doit essayer de changer atomiquement le champ de tous les objets du jeu de données et des invocations de tâches du VDE. Pour chaque objet, le VDE doit renvoyer, dans l'ordre spécifié dans le paramètre Change Scope Objects de la primitive demande, le résultat de sa tentative de changement de champ de cet objet.

NOTE – Le jeu de données et les invocations de tâches, quand elles existent, doivent toujours avoir le même champ d'accès.

The New Scope parameter, of type BOOLEAN, indicates the attempted new scope of access of the objects specified in the request. As defined above, a scope of access must be VDE-specific or VAA-specific. If the scope of access is VAA-specific, the destination VAA name must be also provided (in the New VAA parameter) when it is not the one of the requesting user ; otherwise, it is not mandatory.

The Changed Scope Objects optional parameter, if present, specifies a list of the objects for which the scope must be changed. If this parameter either contains the name of the Data Set or of one or more Task Invocations or is absent, then a change in Executive VAA is required. The Result(+) parameter indicates that the requested service has succeeded.

The List of Change Scope Results parameter contains the result of the change scope operation for each object, in the order specified by the Change Scope Objects parameter of the request primitive. Each element specifies either the success of the change of scope (through the Success parameter), or a reason why the change of scope has failed for this object (in the Change Scope Failure parameter). If the Changed Scope Objects parameter was not present in the request (meaning that only a change in Executive VAA was requested), the List of Change Scope Results shall contain exactly one element which gives the result of the change in Executive VAA.

The Result(-) parameter, of type Service Error, indicates that the service previously requested failed.

8.2.4 Service procedure

On receipt of a ChangeScope request primitive, the DLMS client is identified with its corresponding VAA name. For each submitted name in the Changed Scope Objects parameter, the DLMS server checks the validity of the change of the access and attempts to change the scope of the object. If the Changed Scope Objects parameter was absent, or if it contained the name of the data set or the name of a task invocation, the DLMS server checks the validity of the change of the Executive VAA. If this check is successful, it shall attempt to perform individually the change of scope for all data set and task invocation objects in the VDE. For each object, the VDE shall return, in the order specified in the Changed Scope Objects parameter of the request primitive, the result of its attempt to change the scope of that object.

NOTE – The data set and the task invocation, when they exist, must always have the same scope of access.

9 Services de gestion des invocations de tâches

9.1 Description de l'invocation de tâches

La TI (Task Invocation, invocation de tâches) est un élément qui correspond très exactement à mettre un programme dans la queue d'exécution dans un environnement multitâches. Une TI est associée à une tâche qui contient le code à exécuter ainsi que les variables associées. La TI contient toutes les informations nécessaires à l'initialisation de l'exécution de la tâche correspondante dans le VDE.

Dans DLMS, la TI est un des objets prédéfinis du champ du VDE. La tâche correspondante elle-même peut être chargée avec le contenu d'un jeu de données. Pour plus de détail, voir la spécification d'accompagnement.

9.1.1 L'objet TI

9.1.1.1 Attributs

Les attributs de TI sont :

Object TI
- key attribute : TI Name
- attribute : Scope Of Access (VDE-specific, VAA-specific)
- constraint Scope Of Access = VAA-specific
  - attribute : Executive VAA Name
- attribute : Scope May Change (TRUE, FALSE)
- attribute : Remote Control (TRUE, FALSE)
- attribute : State (UNUSABLE, RUNNING, STOPPED)

Un objet TI est toujours prédéfini dans le VDE. Si l'Executive-VAA existe dans le VDE, le champ d'accès des objets TI est toujours VAA-specific et l'attribut VAA Name spécifie le nom de l'Executive VAA.

9 Task invocation management services

9.1 Task invocation description

The TI is an element which most closely corresponds to put a program on the execution queue in a multi-tasking environment. A TI is associated with a task that contains the code to be executed and the associated variables. The TI contains all the information necessary to initiate the execution of the corresponding task within the VDE.

In DLMS, the TI is one of the predefined objects within the scope of the VDE. The corresponding task itself may be loaded with a data set content. Further details on tasks may be found in companion specifications.

9.1.1 The TI object

9.1.1.1 Attributes

The attributes of a TI are:

Object TI
- key attribute: TI Name
- attribute: Scope Of Access (VDE-specific, VAA-specific)
- constraint Scope Of Access = VAA-specific
  - attribute: Executive VAA Name
- attribute: Scope May Change (TRUE, FALSE)
- attribute: Remote Control (TRUE, FALSE)
- attribute: State (UNUSABLE, RUNNING, STOPPED)

A TI object is always predefined at the VDE. If the Executive VAA exists within the VDE, the scope of access of the TI objects is always VAA-specific and the VAA Name attribute specifies the name of the Executive VAA.

9.1.1.2 Description

L'attribut TI Name identifie la TI de façon unique dans le VDE.

L'attribut Scope Of Access (champ d'accès) précise le champ d'accès actuel de la TI. Ce champ d'accès règle l'accès aux objets. Il doit être VDE-specific ou VAA-specific. S'il y a une Executive VAA dans le VDE, le champ d'accès de tous les objets TI est VAA-specific et le nom de la VAA correspondante est spécifié dans l'attribut Executive VAA Name. Le client DLMS correspondant à cette VAA est le seul autorisé à manipuler les objets TI à l'aide des services DLMS. S'il n'y a pas d'Executive VAA définie dans le VDE, le champ d'accès de tous les objets TI est VDE-specific.

L'attribut Scope May Change (champ peut changer) définit s'il est possible ou non de changer le champ d'accès des objets considérés à l'aide du service ChangeScope de DLMS.

L'attribut Remote Control (contrôle à distance) spécifie si oui ou non la TI peut être établie à distance. Si non (FALSE), la TI ne peut pas être établie à distance en utilisant les services DLMS. Si oui (TRUE), le client DLMS correspondant peut changer l'état de la TI grâce aux services DLMS (moyennant toutes les conditions requises, comme par exemple le Scope Of Access et l'Executive VAA). Dans les deux cas, l'état de la TI peut aussi changer grâce à une action locale. La valeur du paramètre Remote Control elle-même change grâce à une action locale.

L'attribut State (état) indique les principaux états de la TI. Trois états sont définis : STOPPED (arrêtée), RUNNING (active) et UNUSABLE (inutilisable). Avant la première utilisation de la TI, l'attribut State est à STOPPED. Après un démarrage ou une reprise réussis, l'attribut State prend la valeur RUNNING. Dans l'état UNUSABLE, la TI ne peut être ni lancée, ni arrêtée, ni reprise.

Les transitions entre les deux premiers états, STOPPED et RUNNING, ou à partir de l'état UNUSABLE, peuvent être provoquées explicitement via l'utilisation de services DLMS. Les transitions vers l'état UNUSABLE peuvent se produire soit comme résultat d'une action locale (par exemple quand une tâche se suicide), soit comme le résultat indirect de l'utilisation des services DLMS :

- échec non récupérable d'un service de gestion de tâche (Start, Stop, Resume) et

- succès du service Initiate load.

9.1.1.2. Description

The TI Name attribute identifies uniquely the TI within the VDE.

The Scope Of Access attribute defines the current scope of access of the TI. This scope of access rules the Object access. It must be VDE-specific or VAA-specific. If the Executive VAA exists within the VDE, the scope of access of all the TI objects is VAA-specific and the corresponding VAA Name is specified in the Executive VAA Name attribute. The corresponding DLMS client of this VAA is the only one allowed to manipulate the TI objects using DLMS services. If no Executive VAA is defined at the VDE, the scope of access of all the TI objects is VDE-specific.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The Remote Control attribute specifies whether or not the TI may be set under remote control. If false, the state of the TI may not be modified using DLMS services. If true, then the client may change the state of the TI through DLMS services (provided all other necessary conditions, related for instance to the Scope Of Access and the Executive VAA, are fulfilled). In both cases, the state of the TI may change through a local action. The value of the Remote Control attribute itself changes through a local action.

The State attribute indicates the major state of the TI. Three states are defined: STOPPED, RUNNING and UNUSABLE. Before the first use of a TI, the State attribute is set to STOPPED. After a successful start or resume, the State attribute becomes RUNNING. In the UNUSABLE state, the TI may be neither started, stopped, nor resumed.

Transitions between the first two states, STOPPED and RUNNING, or from the UNUSABLE state, may be provoked explicitly through the use of DLMS services. Transitions to the UNUSABLE state may occur either as the result of a local action (for example when a task commits suicide) or as an indirect result of the use of DLMS services:

- unrecoverable failure of a task management service (Start, Stop, Resume), and

- success of the Initiate Load service.

9.1.2 Opérations sur les TI

Cinq services sont disponibles pour le contrôle à distance des TI :

- Start (démarrage) : un client peut utiliser ce service pour passer la TI définie au préalable de l'état STOPPED à l'état RUNNING avec une réinitialisation de la tâche correspondante;

- Stop (arrêt) :un client peut utiliser ce service pour passer la TI de l'état RUNNING à l'état STOPPED;

- Resume (reprise) : un client peut utiliser ce service pour passer la TI de l'état STOPPED à l'état RUNNING sans réinitialiser la tâche correspondante;

- Make Usable (rendre utilisable) : un client peut utiliser ce service pour passer la TI définie au préalable de l'état UNUSABLE à l'état STOPPED;

- GetTIAttribute: procure les attributs courants d'une TI.

9.2 Le service Start *(démarrage)*

9.2.1 Objet

Le service Start permet à un client DLMS de passer une TI à l'état RUNNING. La TI doit être arrêtée pour que la commande soit valable. Avant d'être relancée, la tâche est réinitialisée.

9.2.2 Structure

Tableau 15 - Le service Start

Service Start	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
More Details	U	U(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.2.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service Start.

9.1.2 Operations on the TIs

Five services are used to remote control the TIs:

- Start: a client may use this service to cause a previously defined TI to make a transition from the STOPPED state into the RUNNING state with a reset from the corresponding task;

- Stop: a client may use this service to cause a TI which is in the RUNNING state to make a transition to the STOPPED state;

- Resume: a client may use this service to cause a TI which is in the STOPPED state to make a transition to the RUNNING state without a reset from the corresponding task;

- Make Usable: a client may use this service to cause a previously defined TI to make a transition from the UNUSABLE state into the STOPPED state;

- GetTIAttribute obtains the current attributes of a TI.

9.2 Start service

9.2.1 Purpose

The Start service allows a DLMS client to change the state of a TI to the RUNNING state. The TI must be in the STOPPED state for this service to be valid. Before starting, the task is reset.

9.2.2 Structure

Table 15 - The Start service

Start service	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
More Details	U	U(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.2.3 Parameters

The Argument parameter conveys the specific parameters of the Start service request.

Le paramètre TI Name (nom de la TI), du type ObjectName (nom d'objet), spécifie le nom de la TI à démarrer.

Le paramètre optionnel More Details (plus de détails), du type BIT STRING (chaîne binaire), contient des données supplémentaires pour le démarrage de la TI.

Le paramètre Result(+) indique que le service demandé à fonctionné. A la suite d'un résultat positif, la TI est dans l'état RUNNING. Il n'y a pas d'autres paramètres pour un résultat positif.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'erreur. Le paramètre Error Type est décrit en détail en A.5.

Le paramètre State (état), de type ENUMERATED (énumération), indique l'état de la TI après l'émission de la réponse erreur.

9.2.4 Procédures de service

L'utilisation du service Start doit être conditionnée par des définitions externes des paramètres. L'utilisateur DLMS doit pouvoir identifier à l'aide du paramètre TI Name quelle TI il veut démarrer.

Une réponse erreur est renvoyée dans un des cas suivants :

- le nom d'objet n'existe pas ;

- l'attribut State a la valeur RUNNING ou UNUSABLE ;

- les restrictions d'accès ne sont pas satisfaites.

Les restrictions d'accès sont héritées du champ d'accès de la TI. Les restrictions d'accès ne sont pas satisfaites si le champ d'accès est VAA-specific et si l'Executive VAA n'est pas celle qui a demandé le service.

Si l'on a rencontré des erreurs non récupérables en démarrant la TI, elle peut se mettre dans l'état UNUSABLE et une réponse erreur est renvoyée. Autrement, la TI est réinitialisée, elle passe à l'état RUNNING et tous les paramètres, s'il y en a, sont passés à la tâche. Si la TI utilise un objet jeu de données, son nom est ajouté à l'attribut List of TI du jeu de données. Une réponse positive est alors émise par le serveur DLMS.

The TI Name parameter, of type ObjectName, specifies the name of the TI that is to be started.

The More Details optional parameter, of type BIT STRING, contains the additional data for the TI to start.

The Result(+) parameter indicates that the requested service has succeeded. Following a successful result, the TI is in the RUNNING state. There is no other parameter for the success result.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

The State parameter, of type ENUMERATED, indicates the actual state of the TI after issuing the error response.

9.2.4 Service procedure

Use of the Start service must be dependent on the external definitions of parameters. The DLMS client must identify which TI it wishes to start with the TI Name parameter.

An error response is returned if one of the following statements is met:

- the object name does not exist;

- the state attribute is set to UNUSABLE or RUNNING;

- the access restrictions are not satisfied.

The access restrictions are inherited from the scope of access of the TI. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

If unrecoverable errors are encountered while starting the TI, it may enter in the UNUSABLE state and an Error response is issued. Otherwise, the TI is reset, its state becomes RUNNING and all the parameters, if present, are given to the task. If the TI uses data set objects, its name is added to the List of TI attribute of the data set object. A positive response is then issued by the DLMS server.

9.3 Le service Stop *(arrêt)*

9.3.1 Objet

Le service Stop permet à un client de passer une TI désignée du serveur DLMS de l'état RUNNING à l'état STOPPED.

9.3.2 Structure

Tableau 16 - Le service Stop

Service Stop	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.3.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service Stop.

Le paramètre TI Name (nom de la TI), de type ObjectName (nom d'objet), spécifie le nom de la TI à arrêter.

Le paramètre Result(+) indique que le service demandé a fonctionné. Après un résultat positif, la TI est à l'état STOPPED. Il n'y a pas d'autres paramètres pour un résultat positif.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

Le paramètre State (état), de type ENUMERATED (énumération), indique l'état de la TI après l'émission d'une réponse erreur.

9.3 Stop service

9.3.1 Purpose

The Stop service allows a DLMS client to cause a named TI at the DLMS server to make a transition from the RUNNING state to the STOPPED state.

9.3.2 Structure

Table 16 - The Stop service

Stop service	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.3.3 Parameters

The Argument parameter conveys the specific parameter of the Stop service request.

The TI Name parameter, of type ObjectName, specifies the name of the TI that is to be stopped.

The Result(+) parameter indicates that the requested service has succeeded. Following a successful result, the TI is in the STOPPED state. There is no other parameter for the success result.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

The State parameter, of type ENUMERATED, indicates the actual state of the TI after issuing the error response.

9.3.4 Procédure de service

A la réception d'une indication de service Stop, le serveur DLMS va provoquer le passage de la TI désignée de l'état RUNNING à l'état STOPPED. Le client DLMS doit identifier la TI à arrêter à l'aide du paramètre TI Name.

Une réponse erreur est renvoyée dans un des cas suivants :

- le nom d'objet n'existe pas ;

- l'attribut State a la valeur UNUSABLE ou STOPPED ;

- les restrictions d'accès ne sont pas satisfaites.

Les restrictions d'accès sont héritées du champ d'accès de la TI. Les restrictions d'accès ne sont pas satisfaites si le champ d'accès est VAA-specific et si l'Executive VAA associée n'est pas celle qui a demandé le service.

Si l'on rencontre une erreur irrécupérable en arrêtant une TI, elle peut passer à l'état UNUSABLE et une réponse erreur est émise. Autrement, la TI désignée est passée à l'état STOPPED. Si la TI utilise un objet jeu de données, son nom est retiré de l'attribut List of TI du jeu de données. Une réponse positive est ensuite émise par le serveur DLMS.

9.4 Le service Resume *(reprise)*

9.4.1 Objet

Le service Resume permet à un client DLMS de passer une TI désignée à l'état RUNNING. La TI doit être à l'état STOPPED pour que ce service soit valide. Il n'y a pas de réinitialisation de la TI avant de la remettre dans l'état RUNNING.

9.3.4 Service procedure

On receipt of the Stop service indication, the DLMS server will cause the named TI to make a transition out of the RUNNING state. The DLMS client must identify which TI it wishes to stop with the TI Name parameter.

An error response is returned if one of the following statements is met:

- the object name does not exist;

- the state attribute is set to UNUSABLE or STOPPED;

- the access restrictions are not satisfied.

The access restrictions are inherited from the scope of access of the TI. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

If unrecoverable errors are encountered while stopping the TI, it may enter in the UNUSABLE state and an error response is issued. Otherwise, the named TI is placed in the STOPPED state. If the TI has used data set objects, its name is removed from the List of TI attribute of the data set object. A positive response is then issued by the DLMS server.

9.4 Resume service

9.4.1 Purpose

The Resume service allows a DLMS client to change the state of a named TI to the RUNNING state. The TI must be in the STOPPED state for this service to be valid. No reset of the TI is done before entering in the RUNNING state.

9.4.2 Structure

Tableau 17 - Le service Resume

Service Resume	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.4.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service Resume.

Le paramètre TI Name (nom de la TI), du type ObjectName (nom d'objet), spécifie le nom de la TI à relancer.

Le paramètre Result(+) indique que les service demandé a fonctionné. A la suite d'un résultat positif, la TI est dans l'état RUNNING. Il n'y a pas d'autres paramètres pour un résultat positif.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

Le paramètre State (état), du type ENUMERATED (énumération), indique l'état actuel de la TI après avoir émis une réponse erreur.

9.4.4 Procédure de service

Le client DLMS doit identifier à l'aide du paramètre TI Name quelle TI il souhaite relancer.

Une réponse erreur est renvoyée dans les cas suivants :

- le nom d'objet n'existe pas ;

- l'attribut State a la valeur UNUSABLE ou RUNNING ;

- les restrictions d'accès ne sont pas satisfaites.

9.4.2 Structure

Table 17 - The Resume service

Resume service	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.4.3 Parameters

The Argument parameter conveys the specific parameter of the Resume service request.

The TI Name parameter, of type ObjectName, specifies the name of the TI that is to be resumed.

The Result(+) parameter indicates that the requested service has succeeded. Following a successful result, the TI is in the RUNNING state. There is no other parameter for the success result.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

The State parameter, of type ENUMERATED, indicates the actual state of the TI after issuing the error response.

9.4.4 Service procedure

The DLMS client must identify which TI it wishes to resume with the TI Name parameter.

An error response is returned if one of the following statements is met:

- the object name does not exist;

- the state attribute is set to UNUSABLE or RUNNING;

- the access restrictions are not satisfied.

Les restrictions d'accès sont héritées du champ d'accès de la TI. Les restrictions d'accès ne sont pas satisfaites si le champ est VAA-specific et si l'Executive VAA n'est pas celle qui a demandé le service.

Si l'on a trouvé une erreur irrécupérable en relançant la TI, elle peut se mettre dans l'état UNUSABLE, et une réponse erreur est émise. Autrement, la TI n'est pas réinitialisée et passe à l'état RUNNING. Si la TI utilise un objet jeu de données, son nom est ajouté à l'attribut List of TI de l'objet jeu de données. Une réponse positive est émise par le serveur DLMS.

9.5 Le service MakeUsable *(rendre utilisable)*

9.5.1 Objet

Le service MakeUsable permet à un client DLMS de passer une TI désignée à l'état STOPPED. La TI doit être à l'état UNUSABLE pour que le service soit valide.

Ce service est proposé pour remettre en état les tables de changements d'état.

9.5.2 Structure

Tableau 18 - Le service MakeUsable

Service MakeUsable	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.5.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques à la demande de service MakeUsable.

Le paramètre TI Name (nom de la TI), du type ObjectName (nom d'objet), spécifie le nom de la TI qui doit être rendue utilisable.

The access restrictions are inherited from the scope of access of the TI. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

If unrecoverable errors are encountered while resuming the TI, it may enter the UNUSABLE state and an error response is issued. Otherwise, the TI is not reset and its state becomes RUNNING. If the TI uses data set objects, its name is added to the List of TI attribute of the data set object. A positive response is then issued by the DLMS server.

9.5 MakeUsable service

9.5.1 Purpose

The MakeUsable service allows a DLMS client to change the state of a named TI to the STOPPED state. The TI must be in the UNUSABLE state for this service to be valid.

This service is proposed in order to achieve the state transition table.

9.5.2 Structure

Table 18 - The MakeUsable service

MakeUsable service	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)
State			C	C(=)

9.5.3 Parameters

The Argument parameter conveys the specific parameter of the MakeUsable Service request.

The TI Name parameter, of type ObjectName, specifies the name of the TI that is to be made usable.

Le paramètre Result(+) indique que le service demandé a fonctionné. A la suite d'un résultat positif, la TI est dans l'état STOPPED. Il n'y a pas d'autres paramètres pour un résultat positif.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

Le paramètre State (état), de type ENUMERATED (énumération), indique l'état actuel de la TI après l'envoi du message d'erreur.

9.5.4 Procédure de service

Le client DLMS doit identifier à l'aide du paramètre TI Name quelle TI il veut rendre utilisable.

Une réponse erreur est émise dans les cas suivants :

- le nom d'objet n'existe pas ;

- l'attribut State est à STOPPED ou RUNNING ;

- les restrictions d'accès ne sont pas satisfaites.

Les restrictions d'accès sont héritées du champ d'accès de la TI. Les restrictions d'accès ne sont pas satisfaites si le champ d'accès est VAA-specific et si l'Executive VAA n'est pas celle qui a demandé le service.

Si l'on rencontre une erreur irrécupérable en rendant utilisable la TI, elle restera UNUSABLE et un message d'erreur sera émis. Autrement, la TI est utilisable et passe à l'état STOPPED. Une réponse positive est émise par le serveur DLMS.

The Result(+) parameter indicates that the requested service has succeeded. Following a successful result, the TI is in the STOPPED state. There is no other parameter for the success result.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

The State parameter, of type ENUMERATED, indicates the actual state of the TI after issuing the error response.

9.5.4 Service procedure

The DLMS client must identify what TI it wishes to make usable with the TI Name parameter.

An error response is returned if one of the following statements is met:

- the object name does not exist;

- the state attribute is set to STOPPED or RUNNING;

- the access restrictions are not satisfied.

The access restrictions are inherited from the scope of access of the TI. The access restrictions are not satisfied if the scope of access is VAA-specific and if the associated Executive VAA is not the one that requested the service.

If errors are encountered while making usable the TI, it will stay in the UNUSABLE state and an error response will be issued. Otherwise, the TI is usable and its state becomes STOPPED. A positive response is then issued by the DLMS server.

9.6 Le service GetTIAttribute *(obtenir les attributs de la TI)*

9.6.1 Objet

Le service GetTIAttribute est utilisé pour demander au serveur DLMS tous les attributs associés à une TI désignée.

9.6.2 Structure

Tableau 19 - Le service GetTIAttribute

Service GetTIAttribute	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Scope of Access			M	M(=)
Scope May Change			M	M(=)
Remote Control			M	M(=)
State			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

9.6.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques à la demande de service GetTIAttribute.

Le paramètre TI Name (nom de la TI), du type ObjectName (nom d'objet), spécifie le nom de la TI dont on demande les attributs.

Le paramètre Result(+) indique que le service demandé a fonctionné.

Le paramètre Scope Of Access (champ d'accès), du type ScopeOfAccess, spécifie le champ d'accès de l'objet considéré. Il doit être VDE-specific ou VAA-specific. Si le paramètre Scope of Access contient la valeur VAA-specific, l'objet VAA associé est connu comme étant l'Executive VAA.

Le paramètre Scope May Change (champ peut changer), de type BOOLEAN (booléen), définit si oui (TRUE) ou non (FALSE), le champ d'accès de l'objet considéré peut être modifié à l'aide du service ChangeScope de DLMS.

9.6 GetTIAttribute service

9.6.1 Purpose

The GetTIAttribute service is used to request that a DLMS server returns all the attributes associated with a specified TI.

9.6.2 Structure

Table 19 - The GetTIAttribute service

GetTIAttribute service	Req	Ind	Resp	Conf
Argument	M	M(=)		
TI Name	M	M(=)		
Result(+)			S	S(=)
Scope of Access			M	M(=)
Scope May Change			M	M(=)
Remote Control			M	M(=)
State			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

9.6.3 Parameters

The Argument parameter conveys the specific parameter of the GetTIAttribute service request.

The TI Name parameter, of type ObjectName, specifies the name of the TI of which attributes are requested.

The Result(+) parameter indicates that the requested service has succeeded.

The Scope of Access parameter, of type ScopeOfAccess, specifies the scope of access of the related object. It must be VDE-specific or VAA-specific. If the Scope Of Access parameter contains the VAA-specific value, the associated VAA object name is known to be the Executive VAA.

The Scope May Change parameter, of type BOOLEAN, defines whether (TRUE) or not (FALSE) the scope of access of the related object may change using the DLMS ChangeScope service.

Le paramètre Remote Control (contrôle à distance), de type BOOLEAN (booléen), spécifie si oui (TRUE) ou non (FALSE), la TI peut être contrôlée à distance.

Le paramètre State (état), de type ENUMERATED (énumération), indique l'état de la TI. Trois états sont définis : STOPPED (arrêté), RUNNING (actif) et UNUSABLE (inutilisable). Avant la première utilisation d'une TI, le paramètre State reçoit la valeur STOPPED. Dans l'état UNUSABLE, la TI ne peut être ni démarrée, ni arrêtée, ni relancée.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

9.6.4 Procédure de service

La demande identifie le nom de la TI. Le serveur DLMS vérifie que le nom de la TI est valide; si le champ d'accès de la TI est VAA-specific, il vérifie également que le VAA sur lequel le service est requis est bien l'Executive VAA. Si ces vérifications sont positives, il retourne les attributs définis ci-dessus. Sinon, une réponse négative est retournée.

The Remote Control parameter, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) the TI may be set under remote control.

The State parameter, of type ENUMERATED, indicates the state of the TI. Three states are defined: STOPPED, RUNNING and UNUSABLE. Before the first use of a TI, the State parameter is set to STOPPED. In the UNUSABLE state, the TI may be neither started, stopped nor resumed.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

9.6.4 Service procedure

The request identifies the TI name. The DLMS server checks that the TI name is valid; if the scope of access of the TI is VAA-specific, it also checks that the VAA on which the service is requested is the Executive VAA. If these checks are successful, it returns the attributes defined above. Otherwise, a negative response is returned.

10 Services d'accès aux variables

Les services d'accès aux variables fournissent au client DLMS les moyens permettant d'accéder aux variables typées du VDE. Ces moyens sont fournis par une extension du modèle comprenant trois objets supplémentaires et quatre services qui manipulent ces objets.

10.1 Description du modèle de variables

La variable est un élément abstrait du VDE capable de fournir (quand on la lit) ou d'accepter (quand on l'écrit) la valeur d'une variable typée. Un "type de données" (data type) ou plus simplement "type" est une description abstraite de la classe de données qui peuvent être véhiculées par la valeur de la variable. Le type d'une variable détermine sa syntaxe, l'ordre de grandeur des valeurs possibles, ainsi que sa représentation quand elle est échangée à l'aide de DLMS.

10.1.1 Modèle d'accès aux variables

Ce paragraphe décrit la correspondance entre des objets virtuels appelés variables DLMS et des objets réels (qui ont une réelle existence) appelés objets réels. Un VDE-handler (gestionnaire VDE), qui permet l'accès à une ou plusieurs de ces variables réelles à l'aide des services DLMS d'accès aux variables, doit fournir la correspondance entre les variables réelles auxquelles il est possible d'accéder et un ou plusieurs objets variables DLMS. Cette correspondance est un problème local et ne sera plus discutée par la suite.

DLMS définit trois objets variables qui sont :

- l'objet Variable Désignée ;

- l'objet Boîte à Messages ;

- l'objet Liste de Variables Désignées.

10 Variable access services

The variable access services provide facilities which allow a DLMS client to access typed variables defined at the VDE. These facilities are provided by extending the VDE model with three additional objects and four services which operate upon these objects.

10.1 Variable model description

The variable is an abstract element of the VDE which is capable of providing (when read) or accepting (when written) a typed data value. A "data type" or simply "type" is an abstract description of the class of data which may be conveyed by a variable value. A variable type determines its abstract syntax, its range of possible values, and its representation while being communicated using DLMS.

10.1.1 Variable Access model

This subclause describes the mapping between virtual objects called DLMS variables and real (having a real existence) objects called real variables. A VDE-handler which allows access to one or more of its real variables using the DLMS variable access services shall provide a mapping between the real variables for which access is allowed and one or more DLMS variable access objects. The mapping is a local issue and will not be further discussed here.

DLMS defines three variable access objects which are:

- the Named Variable object;

- the Message Box object;

- the Named Variable List object.

10.1.1.1 Variable Désignée

DLMS définit un objet qui décrit la correspondance entre une variable DLMS et un objet réel au niveau du VDE. C'est l'objet Variable Désignée. L'abstraction représentée par l'objet Variable Désignée modélise la visibilité d'une variable par l'application au niveau VDE.

L'objet Variable Désignée décrit l'accès à une variable réelle en utilisant un nom déterminé par une application. Un objet Variable Désignée peut être utilisé pour décrire un élément de données dépendant d'une application et auquel on accède par un seul nom et par une seule opération.

10.1.1.2 Autres objets variable

La Boîte à Messages est définie comme étant un objet Variable Désignée particulier qui modélise deux files en entrée et en sortie.

L'objet Liste de Variable Désignée est défini comme étant un ensemble explicite d'objets Variables Désignées.

10.1.1.3 Accès à une variable réelle

La correspondance entre un objet Variable Désignée et une variable réelle n'est pas définie dans DLMS. La correspondance doit seulement garantir que l'accès à la variable réelle utilisant cet objet a réussi ou échoué; il n'est pas tenu compte d'un succès partiel.

En plus, le VDE doit garantir que l'accès à une variable DLMS ne peut pas être interrompu. En d'autres termes, la valeur renvoyée après un accès en lecture est cohérente et correspond à un seul instant logique.

NOTES

1 Il est recommandé qu'une variable réelle n'ait qu'une et une seule correspondance avec un objet variable de DLMS.

2 Un objet variable de DLMS n'est jamais créé ou détruit. Son existence est définie dans le VDE ou dans un jeu de données. Un objet variable de DLMS a une durée de vie spécifiée dans le paramètre Lifetime (durée de vie).

10.1.1.1 Named Variable

DLMS defines one object which describes the mapping between a DLMS variable and a real variable at the VDE. It is the Named Variable object. The abstraction represented by the Named Variable object models the application view of a real variable at the VDE.

The Named Variable object describes access to the real variable using an application determined name. A Named Variable object may be used to describe an application-related data element which is accessed using a single name as a single operation.

10.1.1.2 Other variable objects

The Message Box object is defined in DLMS as a particular Named Variable object that models one Input and one Output queue.

The Named Variable List object is defined as an explicit set of Named Variable objects.

10.1.1.3 Access to the real variable

The mapping of a Named Variable object to a real variable is not defined in DLMS. The mapping shall ensure only that access to the real variable using the object either succeeds or fails; partial success is not reported.

Additionally, a VDE shall guarantee that access to a DLMS variable is not interruptive. In other words, the value returned after a read access is coherent and refers to a single logical instant.

NOTES

1 It is recommended that a real variable has one and only one mapping to a DLMS variable object.

2 A DLMS variable object is never created or destroyed. Its existence is defined within the VDE or within the data set. A DLMS variable object has the lifetime specified in its Lifetime parameter.

10.1.1.4 Champ d'accès

Comme pour tous les objets DLMS, l'accessibilité à un objet Variable Désignée, une Boîte à Messages ou une Liste de Variables Désignées est subordonnée à son champ d'accès. Un champ VDE-specific est librement accessible, mais un champ VAA-specific est subordonné à l'existence d'une VAA et on ne peut y accéder que conformément aux restrictions d'accès spécifiques au VAA (VAA-specific).

Pour réussir, un accès à une Liste de Variables Désignées doit satisfaire :

- le champ d'accès de la Liste des Variables Désignées ;

- le champ d'accès de chaque objet appartenant à la liste.

Le succès ou l'échec sont notifiés au niveau des éléments de la liste. L'accès à une Liste de Variables Désignées est équivalent à un accès individuel à chacun des éléments de la liste.

10.1.2 L'objet Variable Désignée

L'objet Variable Désignée décrit la correspondance entre une variable DLMS et une variable réelle définie dans une application au VDE.

10.1.2.1 Attributs

Les attributs de l'objet Variable Désignée sont spécifiés ci-dessous :

Object : Named Variable
key attribute : Variable Name
attribute : Scope Of Access (VDE-specific, VAA-specific)
constraint Scope Of Access = VAA-specific
attribute : VAA Name
attribute : Scope May Change (TRUE, FALSE)
attribute : Lifetime (VDE, DATA-SET)
attribute : Type Description
attribute : Read-Write Flag (READ-ONLY, READ-WRITE)
attribute : Available (TRUE, FALSE)

10.1.2.2 Description

L'attribut Variable Name (nom de variable) identifie de façon unique dans le VDE un objet Variable Désignée. Un Variable Name est le nom d'un objet DLMS ; il est donc défini avec un champ d'accès spécifique.

L'attribut Scope Of Access (champ d'accès) précise le champ d'accès actuel de l'objet désigné. Ce champ d'accès règle l'accès aux objets Variables Désignées. Il doit être VDE-specific ou VAA-specific. Si l'attribut Scope Of Access contient la valeur VAA-specific, le nom de la VAA est spécifié dans l'attribut VAA Name.

10.1.1.4 Scope of access

As all the DLMS objects, the accessibility to a Named Variable object, a Message Box object or a Named Variable List object is subordinated to its scope of access. A VDE-specific scope is freely accessible but a VAA-specific scope is subordinated to the existence of the VAA and must be accessed according to the VAA-specific access restrictions.

To be successful, an access to a Named Variable List object must satisfy:

- the scope of access of the Named Variable List object;

- the scope of access of each Named Variable object member of the list.

Success or failure are reported at the level of the members of the list. Access at a Named Variable List object is equivalent to access at each individual component of the list.

10.1.2 The Named Variable object

The Named Variable object describes the mapping between a DLMS variable and a real application defined variable at the VDE.

10.1.2.1 Attributes

The attributes of a Named Variable object are specified as follows:

Object: Named Variable
key attribute: Variable Name
attribute: Scope Of Access (VDE-specific, VAA-specific)
constraint Scope Of Access = VAA-specific
attribute: VAA Name
attribute: Scope May Change (TRUE, FALSE)
attribute: Lifetime (VDE, DATA-SET)
attribute: Type Description
attribute: Read-Write Flag (READ-ONLY, READ-WRITE)
attribute: Available (TRUE, FALSE)

10.1.2.2 Description

The Variable Name attribute identifies uniquely within the VDE a Named Variable object. A Variable Name is a DLMS object name and is defined with a specific scope of access.

The Scope Of Access attribute specifies the current scope of access for the designated object. This scope of access rules the Named Variable object access. It must be VDE-specific or VAA-specific. If the Scope Of Access attribute contains the VAA-specific value, the name of the VAA is specified in the VAA Name attribute.

L'attribut Scope May Change (champ peut changer) définit si oui ou non le champ d'accès de l'objet considéré peut être modifié à l'aide du service DLMS ChangeScope.

L'attribut Lifetime (durée de vie) spécifie si la durée de vie de l'objet décrit est VDE ou DATA-SET (jeu de données). Un attribut Lifetime de valeur DATA-SET indique que cet objet est défini dans le jeu de données.

L'attribut Type Description (spécification de type) indique le type abstrait de la variable réelle sous-jacente, telle qu'elle est vue en utilisant les services DLMS. Il doit spécifier la classe (Integer8, virgule flottante, ...) et la plage de valeurs pour les éléments de données simples de la variable, ainsi que le regroupement de ces données (en tableaux "array" ou structures), en cas de nécessité, pour les besoins de l'accès. Une variable Type Description et la méthode d'accès doivent fournir l'information suffisante pour permettre au VDE de déterminer la valeur de chaque élément de données de la variable réelle sous-jacente.

L'attribut Read-Write Flag (drapeau lecture-écriture) indique si les droits de lecture et d'écriture de la variable désignée sont READ-WRITE (lecture-écriture) ou READ-ONLY (lecture seulement). Il faut noter qu'une variable désignée peut toujours être lue.

L'attribut Available (disponible) spécifie si la variable réelle sous-jacente est actuellement disponible quand on considère l'accès physique et les ressources matérielles. L'attribut Available et sa gestion dépendent de l'implémentation. Un accès à un objet Variable Désignée dont l'attribut Available a la valeur FALSE (faux) doit renvoyer un message d'erreur d'accès aux données INVALIDATED (non validé).

10.1.2.3 Opérations sur les objets Variables Désignées

DLMS fournit cinq services pour manipuler les objets Variables Désignées :

- Read (lecture) : ce service obtient la valeur actuelle d'une variable réelle décrite par un objet Variable Désignée;

- Write (écriture) : ce service remplace la valeur actuelle de la variable réelle décrite par l'objet Variable Désignée;

- UnconfirmedWrite (écriture non confirmée) : ce service sans confirmation remplace la valeur actuelle de la variable réelle décrite par un objet Variable Désignée;

- InformationReport (rapport d'information) : ce service spontané notifie la valeur actuelle de la variable réelle décrite par un objet Variable Désignée;

- GetVariableAttribute (acquérir les attributs d'une variable) : ce service confirmé obtient la valeur actuelle des attributs d'une ou plusieurs variables.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The Lifetime attribute specifies whether the described object lifetime is VDE or DATA-SET. A Lifetime attribute set to DATA-SET indicates that the object is defined within the data set.

The Type Description attribute indicates the abstract type of the underlying real variable, when viewed using DLMS services. It shall specify the class (Integer8, floating-point, ...) and range of values for the simple data elements of the variable, as well as the grouping of these data elements (into arrays or structures), if applicable, for the purpose of access. A variable Type Description and the access method shall provide sufficient information to the VDE to allow the value of each data element of the underlying real variable to be determined.

The Read-Write Flag attribute indicates whether the Named Variable object read and write right is READ-WRITE or READ-ONLY. Note that a Named Variable object may always be read.

The Available attribute specifies whether or not the underlaying real variable is currently available considering physical access and hardware resources. The Available attribute and its management are implementation dependent. An access to a Named Variable object which Available attribute is set to FALSE shall return an INVALIDATED data access error message.

10.1.2.3 Operations on the Named Variable object

DLMS provides five services to operate upon the Named Variable Object:

- Read: this service obtains the current value of a real variable described by a Named Variable object;

- Write: this service replaces the current value of a real variable described by a Named Variable object;

- UnconfirmedWrite: this unconfirmed service replaces the current value of a real variable described by a Named Variable object;

- InformationReport: this unsolicited service reports the current value of a real variable described by a Named Variable object;

- GetVariableAttribute : this confirmed service obtains the current attributes of one or more variables.

10.1.3 L'objet Boîte à Messages

La Boîte à Messages est un objet Variable Désignée particulier de DLMS qui peut emmagasiner des messages dans le VDE. Il fournit les moyens d'échanger entre utilisateurs des données sans signification pour DLMS.

L'objet Boîte à Messages contient une file FIFO (First In, First Out, premier entré, premier sorti) et les informations concernant cette file. En fonction de la position de son drapeau lecture-écriture, une file peut être lue ou écrite. La façon dont le VDE-handler (gestionnaire VDE) lit ou inscrit les messages dans la file n'est pas dans le domaine de la spécification DLMS. Un objet Boîte à Messages est lu ou écrit à l'aide des services DLMS d'accès aux variables.

10.1.3.1 Attributs

Les attributs de l'objet Boîte à Message sont spécifiés comme suit :

Object : Message Box
key attribute : Message Box Name
attribute : Scope Of Access (VDE-specific, VAA-specific)
constraint Scope Of Access = VAA-specific
Attribute : VAA-Name
attribute : Scope May Change (TRUE, FALSE)
attribute : Lifetime (VDE, DATA-SET)
attribute : Read-Write Flag (READ-ONLY, READ-WRITE)
attribute : Available (TRUE, FALSE)

10.1.3.2 Description

L'attribut Message Box Name (nom de la boîte à messages) identifie de manière unique un objet Variable Désignée. La Boîte à Messages est le nom d'un objet DLMS et il est défini avec un champ d'accès particulier.

L'attribut Scope Of Access (champ d'accès) précise le champ d'accès actuel de l'objet considéré. Ce champ d'accès règle les accès à l'objet Boîte à Messages. Il doit être VDE-specific ou VAA-specific. Si l'attribut Scope Of Access contient la valeur VAA-specific, le nom de la VAA est spécifié dans l'attribut VAA Name.

L'attribut Scope May Change (champ peut changer) définit si oui ou non le champ d'accès de l'objet considéré peut être modifié à l'aide du service ChangeScope de DLMS.

L'attribut Lifetime (durée de vie) spécifie si la durée de vie est VDE ou DATA-SET (jeu de données). Un attribut Lifetime de valeur DATA-SET indique que cet objet est défini dans le jeu de données.

L'attribut Read-Write Flag (drapeau lecture-écriture) indique si les droits de lecture et d'écriture de l'objet Boîte à Messages sont READ-WRITE (lecture-écriture) ou READ-ONLY (écriture seulement). Il faut noter que l'objet Boîte à Messages peut toujours être lu.

10.1.3 The Message Box object

The Message Box object is a specific DLMS Named Variable object that may store messages at the VDE. It provides capabilities for the transmission of any DLMS meaningless data between DLMS users.

The Message Box object contains a First In, First Out (FIFO) queue and information about this queue. Depending on the definition of its Read-Write flag, a queue may be read or written. How the VDE-handler reads or writes a message in the queue is outside the scope of the DLMS specification. A Message Box object is written or read using the DLMS variable access services.

10.1.3.1 Attributes

The attributes of a Message Box object are specified as follows:

Object: Message Box
- key attribute: Message Box Name
- attribute: Scope Of Access (VDE-specific, VAA-specific)
- constraint Scope Of Access = VAA-specific
  - attribute: VAA Name
- attribute: Scope May Change (TRUE, FALSE)
- attribute: Lifetime (VDE, DATA-SET)
- attribute: Read-Write Flag (READ-ONLY, READ-WRITE)
- attribute: Available (TRUE, FALSE)

10.1.3.2 Description

The Message Box Name attribute identifies uniquely a Named Variable object. A Message Box is a DLMS object name and is defined with a specific scope of access.

The Scope Of Access attribute specifies the current scope of access for the designated object. This scope of access rules the Message Box object access. It must be VDE-specific or VAA-specific. If the Scope Of Access attribute contains the VAA-specific value, the name of the VAA is specified in the VAA Name attribute.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The Lifetime attribute specifies whether the described object lifetime is VDE or DATA-SET. A Lifetime attribute set to DATA-SET indicates that the object is defined within the data set.

The Read-Write Flag attribute indicates whether the Message Box object read and write right is READ-WRITE or READ-ONLY. Note that a Message Box object may always be read.

L'attribut Available (disponible) spécifie si oui ou non la variable réelle sous-jacente est actuellement disponible si l'on considère l'accès physique et les ressources matérielles. L'attribut Available et sa gestion dépendent de l'implémentation. Un accès à un objet Boîte à Messages dont l'attribut Available est à FALSE (faux) renvoie un message d'erreur d'accès aux données INVALIDATED (invalidé).

10.1.3.3 Le type Boîte à Messages

La Boîte à Messages est un objet Variable Désignée avec une structure et une sémantique prédéfinies: elle est constituée de deux files d'attentes FIFO, une en lecture et l'autre en écriture. Son type est défini comme décrit ci-dessous :

```
MessageBox_type ::= SEQUENCE {
    queue                   OCTET STRING,
    number_of_messages      Unsigned8,
    available_length        Unsigned16
    }
```

Le champ "queue" (file) contient toujours le numéro du dernier message à transmettre à l'aide des services d'accès aux variables. Si un nouveau message est inscrit, le VDE-handler (gestionnaire VDE) active un mécanisme FIFO qui cache des messages plus anciens du point de vue du client. A cause de ce concept de file d'attente duale, ce champ a un sens différent en lecture et en écriture: elle contient le message en train d'être lu ou écrit. Le VDE-handler doit fournir un mécanisme FIFO pour soit "pousser" les messages déjà écrits, soit pour retirer un nouveau message qui doit être lu. La gestion des files est hors du domaine de cette spécification.

Le champ number_of_messages (nombre de messages) contient le nombre de messages qu'il y a actuellement dans la file de lecture (à lire par le client). Le champ available_length (longueur disponible) exprime en octets l'espace disponible dans la file d'écriture (octets qui peuvent donc être écrits par le client).

10.1.3.4 Opérations sur l'objet Boîte à Messages

DLMS fournit cinq services pour manipuler l'objet Boîte à Messages :

- Read (lecture) : ce service donne la première valeur de la file de lecture, le nombre de messages dans la file (y compris le message en cours de lecture) et la longueur disponible pour l'écriture, comme il est décrit dans le type Boîte à Messages. Par convention, une valeur nulle signifie que le message courant n'est pas significatif (ce cas peut être utilisé pour obtenir, à un instant quelconque, le paramètre longueur disponible);

- Write (écriture) : ce service introduit une nouvelle valeur dans la file d'écriture de l'objet Boîte à Messages. Les valeurs de number_of_messages et de available_length peuvent être présentes dans la demande DLMS, mais elles sont ignorées. Elles sont mises à jour directement par le VDE-handler;

- UnconfirmedWrite (écriture non confirmée) : ce service sans confirmation introduit une nouvelle valeur dans la file de l'objet Boîte à Messages. Les valeurs de number_of_messages et available_length peuvent être présentes dans la demande DLMS, mais elles sont ignorées. Elles sont mises à jour directement par le VDE-handler;

- InformationReport (rapport d'information) : ce service spontané notifie la première valeur de la file de lecture, le nombre de messages dans cette file et la longueur disponible pour l'écriture, comme il est décrit dans le type Boîte à Messages;

The Available attribute specifies whether or not the underlying real variable is currently available considering physical access and hardware resources. The Available attribute and its management are implementation dependent. An access to a Message Box object which Available attribute is set to FALSE shall return an INVALIDATED data access error message.

10.1.3.3 Message Box type

The Message Box is a Named Variable object with a predefined structure and semantics: it is made up of two FIFO queues, one for reading and the other for writing. Its type is fixed and described as follows:

```
MessageBox_type::= SEQUENCE {
        queue                   OCTET STRING,
        number_of_messages      Unsigned8,
        available_length        Unsigned16
        }
```

The queue field always contains the last message that is to be transmitted using DLMS variable access services. If a new message is written, the VDE-handler ensures a FIFO mechanism that hides older messages from the client point of view. Due to the dual-queue concept, this field has a different meaning for Read and for Write operations : it contains the message currently being read or being written. The VDE-handler shall provide a FIFO mechanism to either push the already written messages or to pop a new message to be read. Queue management is outside the scope of this specification.

The number_of_messages field contains the number of messages currently in the Read queue (to be read by the client). The available_length field expresses in bytes the available free space in the Write queue (in bytes) (that may be written by the client).

10.1.3.4 Operations on the Message Box object

DLMS provides five services to operate upon the Message Box object:

- Read: this service obtains the first value of the Read queue, the number of messages in that queue (including the one being read), and the available length left for writing into, as described in the Message Box type. By convention, a value of 0 means that the current message is not significant (this feature can be used to obtain at any time the available_length parameter);

- Write: this service pushes a new value in the Write queue of a Message Box object. The values number_of_messages and available_length may be present in the DLMS request but are ignored. They are updated directly by the VDE-handler;

- UnconfirmedWrite: this unconfirmed service pushes a new value in the queue of a Message Box object. The values number_of_messages and available_length may be present in the DLMS request but are ignored. They are updated directly by the VDE-handler;

- InformationReport: this unsolicited service reports the first value of the Read queue, the number of messages in that queue and the available length for writing, as described in the Message Box type;

- GetVariableAttribute (acquérir les attributs d'une variable): ce service confirmé obtient les attributs courants d'une ou plusieurs variables.

10.1.4 L'objet Liste de Variables Désignées

L'objet Liste de Variables Désignées est fourni comme un moyen pratique de référencer en tant qu'objet unique une liste d'objets Variables Désignées de DLMS indépendants. Chaque élément de la liste doit être un objet Variable Désignée.

L'accès à un élément particulier de la liste est subordonné à l'accès à l'objet Variable Désignée correspondant dans la liste. L'accès utilisant l'objet Liste de Variables Désignées rend compte du succès ou de l'échec pour chaque objet référencé dans la liste. L'accès utilisant l'objet Liste de Variables Désignées est analogue à des accès indépendants aux objets variables référencés dans cette liste.

10.1.4.1 Attributs

Les attributs de l'objet Liste de Variables Désignées sont spécifiés comme suit :

Object : Named Variable List
key attribute : Variable List Name
attribute : Scope Of Access (VDE-specific, VAA-specific)
constraint Scope Of Access = VAA-specific
attribute : VAA Name
attribute : Scope May Change (TRUE, FALSE)
attribute : Lifetime (VDE, DATA-SET)
attribute : List of Named Variable

10.1.4.2 Description

L'attribut Variable List Name (nom de liste de variables) identifie de manière unique l'objet Liste de Variables. Le Variable List Name est un objet DLMS; il est donc défini avec un champ spécifique.

L'attribut Scope Of Access (champ d'accès) précise le champ d'accès actuel de l'objet considéré. Ce champ d'accès règle l'accès à l'objet Liste de Variables Désignées. Il doit être VDE-specific ou VAA-specific. Si l'attribut Scope Of Access contient la valeur VAA-specific, le nom de la VAA est spécifié dans l'attribut VAA Name.

L'attribut Scope May Change (champ peut changer) définit si oui ou non le champ d'accès de l'objet considéré peut être changé à l'aide du service ChangeScope de DLMS.

L'attribut Lifetime (durée de vie) spécifie si la durée de vie de l'objet décrit est VDE ou DATA-SET (jeu de données). Un attribut "Lifetime" dont la valeur est DATA-SET indique que l'objet est défini dans le jeu de données.

L'attribut List of Named Variable (liste de variables désignées) fournit une liste référençant un ou plusieurs objets Variables Désignées.

- GetVariableAttribute: this confirmed service obtains the current attributes of one or more variables.

10.1.4 The Named Variable List object

The Named Variable List object is provided as a convenient way to reference a list of independent DLMS Named Variable objects with an unique object. Each element of the list must be a Named Variable object.

The access to an individual element of the list is subordinated to the access to the corresponding Named Variable object. The access using the Named Variable List object reports success or failure for each object referenced in the list. The access using a Named Variable List object is analogous to independent accesses using the variable objects referenced in the list.

10.1.4.1 Attributes

The attributes of the Named Variable List object are specified as follows:

Object: Named Variable List
key attribute: Variable List Name
attribute: Scope Of Access (VDE-specific, VAA-specific)
constraint Scope Of Access = VAA-specific
attribute: VAA Name
attribute: Scope May Change (TRUE, FALSE)
attribute: Lifetime (VDE, DATA-SET)
attribute: List of Named Variable

10.1.4.2 Description

The Variable List Name attribute identifies uniquely a Named Variable List object. A Variable List Name is a DLMS object and is then defined with a specific scope.

The Scope Of Access attribute defines the current scope of access for the designated object. This scope of access rules the Named Variable List object access. It must be VDE-specific or VAA-specific. If the Scope Of Access attribute contains the VAA-specific value, the name of the VAA is specified in the VAA Name attribute.

The Scope May Change attribute defines whether or not the scope of access of the related object may change using the DLMS ChangeScope service.

The Lifetime attribute specifies whether the described object lifetime is VDE or DATA-SET. A Lifetime attribute set to DATA-SET indicates that the object is defined within the data set.

The List of Named Variable attribute provides a list referencing one or more Named Variable objects.

10.1.4.3 Opérations sur l'objet Liste de Variables Désignées

DLMS fournit cinq services pour manipuler l'objet Liste de Variables Désignées :

- Read (lecture) : ce service obtient la valeur actuelle des variables réelles décrites par les objets Variables Désignées contenus dans l'objet Liste des Variables Désignées;

- Write (écriture) : ce service remplace la valeur actuelle des variables réelles décrites par les objets Variables Désignées contenus dans l'objet Liste de Variables Désignées;

- UnconfirmedWrite : ce service sans confirmation remplace la valeur actuelle des variables réelles décrites par les objets Variables Désignées contenus dans l'objet Liste de Variables Désignées;

- Information report (rapport d'information) : ce service spontané notifie les valeurs actuelles des variables réelles décrites par les objets Variables Désignées contenus dans l'objet Liste de Variables Désignées;

- GetVariableAttribute (acquérir les attributs d'une variable) ce service confirmé obtient les attributs courants d'une ou plusieurs variables.

10.2 Spécification de types

Toutes les variables DLMS sont typées. La description du type d'une variable, telle qu'elle est contenue dans l'attribut Type de l'objet Variable Désignée, fournit une spécification de la syntaxe abstraite et de la plage des valeurs possibles de la variable; elle fournit aussi les bases à partir desquelles on spécifie l'accès détaillé à la variable.

Le type d'une variable peut être simple, spécifiant l'accès à un seul élément de données, ou complexe, spécifiant l'accès à un groupe de types simples combinés. A partir du nom de la variable et de la description de son type, le VDE est capable de localiser chaque élément simple de la variable.

10.1.4.3 Operations on the Named Variable List object

DLMS provides five services to operate upon the Named Variable List object:

- Read: this service obtains the current value of real variables described by the Named Variable objects contained in the Named Variable List object;

- Write: this service replaces the current value of real variables described by the Named Variable objects contained in the Named Variable List object;

- UnconfirmedWrite: this unconfirmed service replaces the current value of real variables described by the Named Variable objects contained in the Named Variable List object;

- InformationReport: this unsolicited service reports the current value of real variables described by the Named Variable objects contained in the Named Variable List object;

- GetVariableAttribute: this confirmed service obtains the current attributes of one or more variables.

10.2 Specification of types

All the DLMS variables are typed. A variable type description, as contained in the Type attribute of the Named Variable object, provides a specification of the abstract syntax and range of possible values of the variable, and also provides the basis upon which detailed access to the variable is specified.

The type of a variable may be simple, specifying access to a single data element, or complex, specifying access to a group of related simple types. From the name of a variable and from its type description, the VDE is able to locate every simple data element of the real variable.

10.2.1 Le paramètre Type Description (spécification de type)

10.2.1.1 Objet

Dans les différents services d'accès aux variables, le type d'une variable est représenté par le paramètre Type Description (spécification de type). La structure de ce paramètre est décrite dans le tableau ci-dessous.

10.2.1.2 Structure

Tableau 20 - Paramètre Type Description

Type Description	Code
Kind of Type	M
Array	S
Number of Elements	M
Type Description	M
Structure	S
List of Type Description	M
Simple	S
Class	M

Comme on peut le voir, un type est décrit par un paramètre spécifié de façon récursive. Ce paramètre décrit une arborescence dite arbre de types. Les feuilles de cet arbre sont des éléments de données simples de la variable décrite par le type. Si un type décrit une variable complexe, l'arbre de types aura alors un ou plusieurs noeuds sans feuilles ; chacun d'eux représente un type complexe constitué à partir des types éventuellement complexes représentés par les noeuds de rang inférieur de l'arbre de types.

10.2.1.3 Paramètres

Le paramètre Kind of Type (genre de type), du type CHOICE (choix), indique les choix effectués pour décrire ce noeud de l'arbre de types. Les valeurs possibles de ce paramètre sont : ARRAY (tableau), STRUCTURE ou SIMPLE.

Le paramètre Array (tableau) est retenu quand le choix pour le paramètre Type Description est ARRAY. Il indique que le noeud décrit est d'un type complexe constitué d'une séquence ordonnée d'éléments tous d'un même type. Le premier élément est numéroté zéro.

Le paramètre Number of Elements (nombre d'éléments), du type Unsigned8, spécifie le nombre d'éléments dans le tableau.

Le paramètre Type Description (spécification de type) spécifie le type des éléments du tableau par une référence récursive au paramètre Type Description.

10.2.1 Type Description parameter

10.2.1.1 Purpose

In the various variable access services, the type of a variable is represented by the Type Description parameter. The structure of this parameter is described in the following table.

10.2.1.2 Structure

Table 20 - Type Description parameter

Type Description	Code
Kind of Type	M
Array	S
Number of Elements	M
Type Description	M
Structure	S
List of Type Description	M
Simple	S
Class	M

As can be seen, a type is described by a recursively specified parameter. This parameter describes a branching tree, called a type tree. The leaves of this tree are the simple data elements of the variable described by the type. If a type describes a complex variable, then the type tree will have one or more non-leaf nodes, each of which represents a complex type constructed from the possible complex types represented by the subordinate nodes of the type tree.

10.2.1.3 Parameters

The Kind of Type parameter, of type CHOICE, indicates the selection which has been chosen to describe this node of the type tree. The possible values for this parameter are: ARRAY, STRUCTURE or SIMPLE.

The Array parameter is selected when the selection for the Type Description parameter is ARRAY. It indicates that the node being described is a complex type that is constructed from an ordered sequence of elements, all of a single type. The first element is numbered zero.

The Number of Elements parameter, of type Unsigned8, specifies the number of elements in the array.

The Type Description parameter specifies the type of the array elements through a recursive reference to the Type Description parameter.

Le paramètre Structure est retenu quand le choix pour le paramètre Type Description est STRUCTURE. Il indique que le noeud décrit est d'un type complexe, qu'il est constitué par une liste ordonnée de un ou plusieurs composants, chacun d'eux pouvant avoir un type distinct. Les composants de la structure sont ordonnés, le premier a le numéro zéro.

Le paramètre List of Type Description (liste de spécifications de types) décrit les composants de la structure en terme de types (les possibilités de type pour le type choice). C'est une référence récursive à Type Description.

Le paramètre Simple est retenu quand le choix pour le paramètre Type Description est SIMPLE. Il indique que l'on décrit une feuille de l'arbre de types. Un tel noeud contient la classe de l'élément de données représenté par ce noeud.

Le paramètre Class (classe) spécifie la classe des éléments de données représentés par les feuilles. La valeur de ce paramètre est l'une des suivantes : BOOLEAN (booleen), INTEGER (nombre entier), LONG (long), DOUBLE-LONG (double longueur), UNSIGNED (sans signe), LONG-UNSIGNED (long sans signe), DOUBLE-LONG-UNSIGNED (double longueur sans signe), BIT-STRING (chaîne binaire), OCTET-STRING (chaîne d'octets), VISIBLE-STRING (chaîne en clair), FLOATING-POINT (nombre à virgule flottante), TIME (heure) et BCD (caractères). Les classes possibles de ce paramètre sont totalement compatibles avec la norme ASN.1 (ISO/CEI 8824).

10.2.2 Le paramètre Detailed Access (accès détaillé)

10.2.2.1 Objet

Le type d'une variable décrit la syntaxe abstraite et la plage des valeurs possibles de la variable réelle dans le VDE. La description de l'accès détaillé spécifie une vue détaillée d'une variable de ce type. Il peut être utilisé pour restreindre l'accès à un sous ensemble de la plage des valeurs possibles de la variable. L'accès détaillé établit une correspondance entre la vue fournie par l'objet considéré et celle qui est souhaitée pour l'accès. Le résultat est une correspondance indirecte avec la variable réelle.

Dans les divers services d'accès aux variables, l'accès détaillé est représenté par le paramètre Detailed Access.

The Structure parameter is selected when the selection for the Type Description parameter is STRUCTURE. It indicates that the node being described is a complex type that is constructed from an ordered list of one or more components, each of which may have a distinct type. The components of the structure are ordered and the first component is numbered zero.

The List of Type Description parameter describes the components of the structure in terms of types (the type possibilities for the choice type). This is a recursive reference to the Type Description.

The Simple parameter is selected when the selection for the Type Description parameter is SIMPLE. It indicates that a leaf node of the type tree is being described. Such a node contains the class of the data element represented by the node.

The Class parameter specifies the class of the data element represented by the leaf node. The value of this parameter is chosen from: BOOLEAN, INTEGER, LONG, DOUBLE-LONG, UNSIGNED, LONG-UNSIGNED, DOUBLE-LONG-UNSIGNED, BIT-STRING, OCTET-STRING, VISIBLE-STRING, FLOATING-POINT, TIME or BCD. The possible Class parameter values are fully compatible with the ASN.1 standard (ISO/IEC 8824).

10.2.2 Detailed Access parameter

10.2.2.1 Purpose

A type of variable describes the abstract syntax and range of possible values of the real variable in the VDE. The detailed access description specifies a detailed view of a variable of this type. It may be used to restrict access to a subset of the range of possible values of the variable. Detailed access to a variable provides a mapping from the view provided by the referenced object to the view which is desired by the access. This results in an indirect mapping to the real variable.

In the various variable access services, detailed access is represented by the presence of the Detailed Access parameter.

La description du paramètre Detailed Access repose sur ses relations avec le type des variables DLMS. A la suite de cette description, on décrira les relations entre le paramètre Detailed Access et la mise en correspondance fournie par l'objet variable.

10.2.2.2 Structure

Tableau 21 - Le paramètre Detailed Access

Detailed Access	Code
List of Subelement	M
Detailed Access Selected	S
Access Selection	M
Component	S
Index	S
Index Range	S
Start Index	M
Number of Index	M
Detailed Access	M
Access Selected	S
Component	S
Index	S
Index Range	S
Start Index	M
Number of Index	M

10.2.2.3 Paramètres

Le paramètre List of Subelements (liste de sous-éléments) spécifie une liste contenant un ou plusieurs paramètres Subelement. Chaque paramètre Subelement sélectionne un noeud ou, dans le cas d'un tableau (array), une plage de noeuds, du niveau d'imbrication immédiatement supérieur dans l'arbre de types. Ce sous-élément peut servir à une spécification supplémentaire d'accès détaillé ou à la spécification d'accès aux éléments de données représentés par les noeuds sélectionnés.

Quand le paramètre List of Subelements contient plus d'un élément, le type déduit de ce paramètre est une structure. Les composants de ce type déduit ont des types déterminés par les sous-éléments spécifiés dans cette liste. Quand le paramètre List of Subelements contient un seul élément, le type déduit de ce paramètre est déterminé par le sous-élément spécifié pour lui.

Le paramètre Detailed Access Selected (accès détaillé sélectionné) spécifie qu'une branche au niveau d'imbrication immédiatement supérieur de l'arbre de types a été sélectionnée pour un accès détaillé récursif.

Le paramètre Access Selection (sélection de l'accès), du type CHOICE (choix), indique quel genre d'accès a été sélectionné. Les valeurs possibles sont : COMPONENT (composant), INDEX (index), INDEX-RANGE (plage d'index). COMPONENT sélectionne un seul composant de la structure tel qu'identifié par le paramètre Component. INDEX sélectionne un seul élément d'un tableau (array) comme spécifié par le paramètre Index. INDEX-RANGE sélectionne un tableau d'éléments comme spécifié par le paramètre Index Range.

The description of the Detailed Access parameter is based upon its relationship to a DLMS variable type. Following this description, the relationship between the Detailed Access parameter and the mapping provided by a variable object will be described.

10.2.2.2 Structure

Table 21 - Detailed Access parameter

Detailed Access	Code
List of Subelement	M
Detailed Access Selected	S
Access Selection	M
Component	S
Index	S
Index Range	S
Start Index	M
Number of Index	M
Detailed Access	M
Access Selected	S
Component	S
Index	S
Index Range	S
Start Index	M
Number of Index	M

10.2.2.3 Parameters

The List of Subelement parameter specifies a list containing one or more Subelement parameters. Each Subelement parameter selects a node or, in the case of array, a range of nodes at the next higher nesting level of the type tree. This Subelement may be for the purpose of additional detailed access specification, or for specifying access to the data elements represented by the selected nodes.

When the List of Subelement parameter contains more than one element, the derived type resulting from this parameter is a structure. The components of this derived type have types as determined by the subelements specified in the list. When the List of Subelement parameter contains one element, the derived type resulting from this parameter is determined by the subelement specified for it.

The Detailed Access Selected parameter specifies that one subtree at the next higher nesting level of the type tree is selected for recursive detailed access.

The Access Selection parameter, of type CHOICE, indicates which of the access kind is selected. The possible values are COMPONENT, INDEX or INDEX-RANGE. COMPONENT selects a single component of the structure as identified by the Component parameter. INDEX selects a single array element, as specified by the Index parameter. INDEX-RANGE selects an array of elements, as specified by the Index Range parameter.

Le paramètre Component (composant), de type Unsigned8, est sélectionné si le noeud actuel de l'arbre de types spécifie une structure et que le paramètre Access Selection indique COMPONENT. Le type dérivé résultant de l'application du paramètre Component Selection est déterminé par l'application du paramètre Detailed Access au noeud sélectionné de l'arbre de types. Si le paramètre Detailed Access Selected est retenu, le composant doit être un tableau (array) ou une structure.

Le paramètre Index, de type Unsigned8, est sélectionné si le noeud actuel de l'arbre de types spécifie un tableau (array) et si le paramètre Access Selection indique INDEX. Il sélectionne un élément spécifique du tableau. Si le paramètre Detailed Access Selected est retenu, cet élément spécifique du tableau est lui-même un tableau ou une structure. Le type dérivé résultant de l'application du paramètre Component Selection est déterminé par l'application du paramètre Detailed Access au noeud sélectionné de l'arbre de types.

Le paramètre Index Range (plage d'index), est sélectionné si le noeud actuel de l'arbre de types spécifie un tableau et si le paramètre Access Selection indique INDEX-RANGE. Il sélectionne une plage d'éléments du tableau. Si le paramètre Detailed Access Selected est retenu, l'élément spécifique du tableau doit être lui-même un tableau ou une structure. Le type dérivé résultant de l'application du paramètre Component Selection est déterminé par l'application du paramètre Detailed Access au noeud sélectionné de l'arbre de types.

Le paramètre Start Index (début d'index), de type Unsigned8, indique le début de la plage d'index. Ce doit être un index valide du tableau. L'élément spécifié est le premier élément du tableau dérivé, et il est numéroté zéro dans ce type.

Le paramètre Number of Index (nombre d'index), de type Unsigned8, indique le nombre d'éléments à inclure dans le tableau dérivé, y compris l'élément sélectionné par le paramètre Start Index. Si le paramètre a la valeur zéro, tous les éléments du tableau ayant un index supérieur à la valeur du paramètre Start Index sont sélectionnés.

Le paramètre Detailed Access (accès détaillé) spécifie des accès détaillés supplémentaires au niveau du noeud sélectionné.

Le paramètre Access Selected (accès sélectionné), du type CHOICE (choix), spécifie que l'une des branches du noeud de niveau immédiatement supérieur de l'arbre de types est sélectionnée pour un accès. Quant au paramètre Access Selection, ses valeurs possibles sont les suivantes : COMPONENT (composant), INDEX (index) ou INDEX-RANGE (plage d'index).

Les paramètres du paramètre Access Selected sont identiques à ceux décrits pour le paramètre Detailed Access Selected. Se référer à la description ci-dessus.

10.2.2.4 Conformité

Le niveau d'accès détaillé supporté par le VDE dépend de deux bits du descripteur d'accès détaillé dans le Bloc de Conformité. Le descripteur d'accès détaillé est décrit en détail en 6.1.

The Component parameter, of type Unsigned8, is selected if the current node of the type tree specifies a structure and the Access Selection parameter indicates COMPONENT. The derived type which results from applying the Component Selection parameter is determined by application of the Detailed Access parameter to the selected type tree node. If the Detailed Access Selected parameter is selected, the component shall be an array or a structure.

The Index parameter, of type Unsigned8, is selected if the current node of the type tree specifies an array and the Access Selection parameter indicates INDEX. It selects the specific array element. If the Detailed Access Selected parameter is selected, the specific element of the array shall be itself an array or a structure. The derived type which results from applying the Component Selection parameter is determined by application of the Detailed Access parameter to the selected type tree node.

The Index Range parameter is selected if the current node of the type tree specifies an array and the Access Selection parameter indicates INDEX-RANGE. It selects a range of array elements. If the Detailed Access Selected parameter is selected, the specific element of the array shall be itself an array or a structure. The derived type which results from applying the Component Selection parameter is determined by application of the Detailed Access parameter to the selected type tree node.

The Start Index parameter, of type Unsigned8, indicates the start of the index range. It must be a valid index of the array. The specified element is the first element of the resulting derived array and is numbered zero in that type.

The Number of Index parameter, of type Unsigned8, indicates the number of elements to be included in the derived array, including the element selected by the Start Index parameter. If the parameter has the value zero, all the elements of the array with an index greater than the Start Index parameter value are selected.

The Detailed Access parameter specifies additional detailed access specification at the selected node.

The Access Selected parameter, of type CHOICE, specifies that one subtree at the next higher nesting level of the type tree is selected for access. As for the Access Selection parameter, the possible values are COMPONENT, INDEX or INDEX-RANGE.

The parameters of the Access Selected parameter are identical to those described for the Detailed Access Selected parameter. Please refer above for their description.

10.2.2.4 Conformance

The level of the detailed access supported at a VDE depends on the two bits of the Detailed Access descriptor of the Conformance Block. The Detailed Access descriptor is described in detail in 6.1.

10.3 Spécification de la valeur des données

Les services d'accès aux variables sont utilisés pour obtenir ou pour mettre à jour la valeur de une ou plusieurs variables DLMS référencées par les objets Variable Désignée ou Liste de Variables Désignées. Ces valeurs sont véhiculées à l'aide du paramètre Data (données). Les erreurs d'accès sont spécifiées dans le paramètre Data Access Error.

10.3.1 Le Paramètre Data

10.3.1.1 Objet

Le paramètre Data est utilisé par les services Read, Write, UnconfirmedWrite et InformationReport pour véhiculer la valeur de la variable.

10.3.1.2 Structure

Tableau 22 - Le paramètre Data

Data	Code
Kind of Data	M
Array	S
List of Data	M
Structure	S
List of Data	M
Simple	S
Class	M
Value	M
CompactArray	S
Contents Description	M
Array Contents	M

Comme il est indiqué dans le tableau, le paramètre Data est défini de façon récursive. Il spécifie une arborescence ; chaque noeud de cet arbre correspond dans une relation un pour un à un noeud de l'arbre de types ou à celui de l'arbre de types dérivé après l'application de l'accès détaillé.

10.3.1.3 Paramètres

Le paramètre Kind of Data (genre de données), du type CHOICE (choix) identifie le genre de données contenues dans le noeud actuel. Les valeurs possibles de ce paramètre sont : ARRAY (tableau), STRUCTURE, SIMPLE ou COMPACT-ARRAY (tableau compact).

Le paramètre Array est sélectionné quand le choix du paramètre Kind of Data est ARRAY. Il fournit une liste ordonnée de paramètre Data. Chaque élément de cette liste fournit la valeur de l'élément correspondant du tableau.

10.3 Specification of data values

The DLMS variable access services are used to obtain or to update the value of one or more DLMS variables referenced by Named Variable or Named Variable List objects. Those values are conveyed using the Data parameter. The access errors are specified with the Data Access Error parameter.

10.3.1 Data parameter

10.3.1.1 Purpose

The Data parameter is used by the Read, Write, UnconfirmedWrite and InformationReport services to convey the value of a variable.

10.3.1.2 Structure

Table 22 - Data parameter

Data	Code
Kind of Data	M
Array	S
List of Data	M
Structure	S
List of Data	M
Simple	S
Class	M
Value	M
CompactArray	S
Contents Description	M
Array Contents	M

As stated in the table, the Data parameter is recursively defined. It specifies a branching tree. Each node of this tree corresponds in a one-to-one relation to a node of the variable type tree or derived type tree after applying detailed access.

10.3.1.3 Parameters

The Kind of Data parameter, of type CHOICE, identifies the kind of data contained in the current node. The possible values for this parameter are: ARRAY, STRUCTURE, SIMPLE or COMPACT-ARRAY.

The Array parameter is selected when the selection for the Kind of Data parameter is ARRAY. It provides an ordered list of Data parameters. Each element of this list provides the value of the corresponding element of the array.

Le paramètre List of Data (liste de données) fournit des spécifications supplémentaires des données au noeud sélectionné.

Le paramètre Structure est retenu quand le choix du paramètre Kind of Data est STRUCTURE. Il fournit une liste ordonnée de paramètres Data, chacun d'eux pouvant avoir un type différent. Chaque élément de la liste fournit la valeur de l'élément correspondant de la structure.

Le paramètre List of Data (liste de données) fournit des spécifications supplémentaires des données au noeud sélectionné.

Le paramètre Simple est retenu quand le choix du paramètre Kind of Data est SIMPLE. Il indique que la description porte sur une feuille de l'arbre de données. Un tel noeud contient la classe de l'élément de données simple de la variable ainsi que sa valeur.

Le paramètre Class précise la classe d'élément de données représentée par le noeud. La valeur de ce paramètre est choisie parmi : BOOLEAN (booleen), INTEGER (nombre entier), LONG (long), DOUBLE-LONG (double longueur), UNSIGNED (sans signe), LONG-UNSIGNED (long sans signe), DOUBLE-LONG-UNSIGNED (double longueur sans signe), BIT-STRING (chaîne binaire), OCTET-STRING (chaîne d'octets), VISIBLE-STRING (chaîne en clair), FLOATING-POINT (nombre à virgule flottante), TIME (heure) et BCD (caractères).

Le paramètre Value (valeur) contient la valeur actuelle de l'élément de données simple conforme à la valeur du paramètre Class associé.

Le paramètre CompactArray est sélectionné quand le choix du paramètre Kind of Data est COMPACT-ARRAY. Il contient une description d'un type DLMS, et de données constituées d'éléments de données de même type.

Le paramètre Contents Description, de type Type Description, donne le type de chacun des éléments de données.

Le paramètre Array Contents, de type OCTET-STRING, contient tous les éléments de données dans une forme compacte dérivée du paramètre Contents Description.

10.3.2 Le paramètre Data Access Error (erreur d'accès aux données)

Le paramètre Data Access Error (erreur d'accès aux données), du type ENUMERATED (énumération), indique la raison de l'erreur lors de la tentative d'accès à une variable. Les valeurs possibles du paramètre sont: SCOPE-OF-ACCESS-VIOLATED (champ d'accès violé), OBJECT-UNAVAILABLE (objet indisponible), HARDWARE-FAULT (erreur matériel), TEMPORARY-FAILURE (erreur temporaire), OBJECT-UNDEFINED (objet indéfini), OBJECT-CLASS-INCONSISTENT (classe d'objet sans signification), TYPE-UNMATCHED (type ne correspond pas), ou READ-WRITE-DENIED (protection lecture-écriture).

The List of Data parameter specifies additional data specification at the selected node.

The Structure parameter is selected when the selection for the Kind of Data parameter is STRUCTURE. It provides an ordered list of Data parameters, each of which may have a distinct type. Each element of this list provides the value of the corresponding element of the structure.

The List of Data parameter specifies additional data specification at the selected node.

The Simple parameter is selected when the selection for the Kind of Data parameter is SIMPLE. It indicates that a leaf node of the data tree is being described. Such a node contains the class of the simple data element of the variable and its value.

The Class parameter specifies the class of the data element represented by the leaf node. The value of this parameter is chosen from: BOOLEAN, INTEGER, LONG, DOUBLE-LONG, UNSIGNED, LONG-UNSIGNED, DOUBLE-LONG-UNSIGNED, BIT-STRING, OCTET-STRING, VISIBLE-STRING, FLOATING-POINT, TIME or BCD.

The Value parameter contains the actual value of the simple data element according to the associated Class parameter value.

The CompactArray parameter is selected when the selection for the Kind of Data parameter is COMPACT-ARRAY. It contains a description of a DLMS type, and data consisting of identically typed data elements.

The Contents Description parameter, of type Type Description, gives the type of each individual data element.

The Array Contents parameter, of type OCTET-STRING, contains all the data elements in compact form derived from the Contents Description parameter.

10.3.2 Data Access Error parameter

The Data Access Error parameter, of type ENUMERATED, indicates the reason for the failure of an attempted access to a variable. The possible values for the parameter are: SCOPE-OF-ACCESS-VIOLATED, OBJECT-UNAVAILABLE, HARDWARE-FAULT, TEMPORARY-FAILURE, OBJECT-UNDEFINED, OBJECT-CLASS-INCONSISTENT, TYPE-UNMATCHED or READ-WRITE-DENIED.

Le code erreur SCOPE-OF-ACCESS-VIOLATED notifie que l'accès tenté à une variable ne respecte pas le champ d'accès de la variable. Le code erreur OBJECT-UNAVAILABE notifie que l'accès tenté vise une variable non validée actuellement. Le code erreur HARDWARE-FAULT notifie une erreur physique de longue durée de l'équipement. Le code erreur TEMPORARY-FAILURE notifie une erreur physique temporaire de l'équipement. Le code erreur OBJECT-UNDEFINED notifie que l'objet est non défini dans le VDE. Le code d'erreur OBJECT-CLASS-INCONSISTENT notifie que la classe de l'objet est incompatible avec le service demandé. Le code erreur TYPE-UNMATCHED notifie que l'accès tenté vise une variable dont le type est incompatible avec le type de variable. Le code d'erreur READ-WRITE-DENIED notifie que le drapeau de lecture-écriture Read-Write de la variable est incompatible avec le service demandé.

Le paramètre Data Access Error n'indique pas qu'il y a une erreur dans la demande de service. Il indique une erreur de la tentative d'accès à une variable au niveau de la variable. Lors de l'accès à un objet Liste de Variables Désignées, il notifie une erreur d'accès à un des éléments de la liste. Dans le cas d'un accès détaillé, il n'est pas fait mention des résultats des autres accès demandés à la même variable.

10.3.3 Le paramètre Variable Access Specification (spécification d'accès à une variable)

10.3.3.1 Objet

Le présent paragraphe traite de paramètres qui spécifient l'accès à une seule variable. Il n'explique pas la correspondance d'une variable avec une variable réelle. Il décrit les paramètres que les services d'accès aux variables utilisent pour repérer les variables réelles.

10.3.3.2 Structure

Tableau 23 - Paramètre Variable Access Specification

Variable Access Specification	Code
Kind of Access	M
Variable Name	S
Detailed Variable	S
Variable Name	M
Detailed Access	M

10.3.3.3 Paramètres

Le paramètre Kind of Access (genre d'accès), du type CHOICE (choix), indique si l'accès est spécifié en termes de noms de variable ou d'accès détaillé. Les valeurs possibles de ce paramètre sont : VARIABLE-NAME (nom de variable) ou DETAILED-ACCESS (accès détaillé).

Le paramètre Variable Name (nom de variable), du type ObjectName (nom d'objet), est spécifié quand l'accès VARIABLE-NAME est sélectionné. Il identifie de manière unique dans le VDE l'objet Variable Désignée ou l'objet Liste de Variables Désignées auquel il faut accéder.

The SCOPE-OF-ACCESS-VIOLATED error reports an attempted access to a variable that violates the scope of access of the variable. The OBJECT-UNAVAILABLE error reports an attempted access to a variable not currently validated. The HARDWARE-FAULT error reports a physical error of long duration from the hardware. The TEMPORARY-FAILURE error reports a temporarily physical error from the hardware. The OBJECT-UNDEFINED error reports that the object is not defined in the VDE. The OBJECT-CLASS-INCONSISTENT error reports that the class of object is incompatible with the asked service. The TYPE-UNMATCHED error reports an attempted access to a variable using a type incompatible with the variable type. The READ-WRITE-DENIED error reports that the Read-Write flag of the variable is incompatible with the asked service.

The Data Access Error parameter does not indicate failure of a service request. It indicates a failure of an attempted variable access at the level of the variable. In a Named Variable List object access, it specifies an access error to a member of the list. For a detailed access, no report is made on the result of other requested accesses for the same variable.

10.3.3 Variable Access Specification parameter

10.3.3.1 Purpose

This subclause reports parameters which specify the access to a single variable. It does not explain the mapping of a variable on a real variable. It describes the parameters that the variable access services use to point out the real variable.

10.3.3.2 Structure

Table 23 - Variable Access Specification parameter

Variable Access Specification	Code
Kind of Access	M
Variable Name	S
Detailed Variable	S
Variable Name	M
Detailed Access	M

10.3.3.3 Parameters

The Kind of Access parameter, of type CHOICE, indicates whether the access is specified in terms of a variable name or a variable detailed access. The possible values for the parameter are: VARIABLE-NAME or DETAILED-ACCESS.

The Variable Name parameter, of type ObjectName, is specified when the VARIABLE-NAME access is selected. It identifies uniquely within the VDE the Named Variable or Named Variable List object to be accessed.

Le paramètre Detailed Variable (variable détaillée) est retenu quand le paramètre Kind of Access est sur DETAILED-ACCESS. Il indique la variable à laquelle il faut accéder conformément à l'accès détaillé spécifié.

Dans ce cas, le paramètre Variable Name, du type ObjectName, identifie de manière unique dans le VDE l'objet Variable Désignée auquel on accède avec un accès détaillé. Et le paramètre Detailed Access indique l'accès détaillé qui s'applique à cette occurrence du service. Ce paramètre est décrit ci-dessus en 10.2.

10.4 Le service Read *(lecture)*

10.4.1 Objet

Le service Read est utilisé par un client DLMS pour demander au VDE de renvoyer la valeur et la description du type d'une ou de plusieurs variables définies dans le VDE.

10.4.2 Structure

Tableau 24 - Le service Read

Service Read	Req	Ind	Resp	Conf
Argument	M	M(=)		
List of Variable Access Specification	M	M(=)		
Result(+)			S	S(=)
List of Read Result			M	M(=)
Data			S	S(=)
Data Access Error			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.4.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service Read.

Le paramètre List of Variable Access Specification (spécification d'une liste de variables à accéder) spécifie une ou plusieurs variables auxquelles il faut accéder. Le paramètre Variable Access Specification est décrit ci-dessus en 10.3.3.

Le paramètre Result(+) indique que le service demandé a fonctionné.

The Detailed Variable parameter is selected when the Kind of Access parameter is set to DETAILED-ACCESS. It indicates the variable to be accessed according to specified detailed accesses.

In this case, the Variable Name parameter, of type ObjectName, identifies uniquely within the VDE the Named Variable object to be accessed with a detailed access. And the Detailed Access parameter indicates the detailed access that applies to this service instance. This parameter is described in 10.2 above.

10.4 Read service

10.4.1 Purpose

The Read service is used by a DLMS client in order to request the VDE to return the value and the type description of one or more variables defined at the VDE.

10.4.2 Structure

Table 24 - The Read service

Read service	Req	Ind	Resp	Conf
Argument	M	M(=)		
List of Variable Access Specification	M	M(=)		
Result(+)			S	S(=)
List of Read Result			M	M(=)
Data			S	S(=)
Data Access Error			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.4.3 Parameters

The Argument parameter conveys the specific parameter of the Read Service request.

The List of Variable Access Specification parameter specifies one or more variables which are to be accessed. The Variable Access Specification parameter is described above in 10.3.3.

The Result(+) parameter indicates that the requested service has succeeded.

Le paramètre List of Read Result (liste des résultats des lectures) contient les valeurs des variables spécifiées, dans l'ordre défini par le paramètre List of Variable Access Specification de la primitive de demande. Chaque élément spécifie soit la valeur de la variable réelle au moment de l'accès, soit la raison pour laquelle la lecture a échoué pour cette variable.

Le paramètre Data (données), du type Data (données), contient la valeur de la variable. La syntaxe abstraite de la valeur de la donnée est déterminée par le type dérivé spécifié par l'accès (quand on utilise l'accès détaillé), ou par le type défini pour une variable (quand on utilise l'accès direct). La description détaillée de ce paramètre est fournie en 10.3.1.

Le paramètre Data Access Error (erreur d'accès aux données), du type DataAccessError, fournit la raison de l'erreur de lecture. La description détaillée de ce paramètre est fournie en 10.3.2.

Le paramètre Result(-) indique que le service demandé au préalable a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

10.4.4 Procédure de service

Pour le service Read, un résultat positif signifie que la demande de service était acceptable par le VDE et que le VDE a tenté de lire les valeurs de chacune des variables spécifiées dans la demande.

Après avoir vérifié la validité de la demande de service, le VDE essaie de lire les valeurs des variables spécifiées. Il renvoie pour chaque tentative de lecture, soit une valeur, soit la raison de l'échec, dans l'ordre spécifié par le paramètre List of Variable Access Specification (spécification d'une liste de variables à accéder) de la primitive de demande.

10.5 Le service Write *(écriture)*

10.5.1 Objet

Le service Write est utilisé par un client DLMS pour demander au VDE de remplacer le contenu d'une ou plusieurs variables par les valeurs fournies dans la demande.

NOTE – La nouvelle valeur de la variable spécifiée dans la demande d'écriture doit correspondre au type défini de la Variable Désignée. Un nouveau type ne peut pas être défini dynamiquement dans DLMS.

The List of Read Result parameter contains the values of the specified variables, in the order specified by the List of Variable Access Specification parameter of the request primitive. Each element specifies either the value of the real variable at the time of access or a reason for the read to fail for this variable.

The Data parameter, of type Data, contains the value of the variable. The abstract syntax of a data value is determined by the derived type specified by the access (when using detailed access) or by the defined type of a variable (when using direct access). The description of this parameter is detailed in 10.3.1.

The Data Access Error parameter, of type DataAccessError, provides the reason for the read to fail. The description of this parameter is detailed in 10.3.2.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

10.4.4 Service procedure

For the Read service, a successful result means that the service request was acceptable for the VDE and that the VDE has attempted to read the value of each of the specified variables in the request.

After verifying the validity of the service request, the VDE attempts to read the values of the specified variables. It returns for each read attempt, in the order specified in the List of Variable Access Specification parameter of the request primitive, a value or a reason for the failure.

10.5 Write service

10.5.1 Purpose

The Write service is used by a DLMS client to request the VDE to replace the content of one or more variables with values supplied in the request.

NOTE – The variable new value specified in the Write request must correspond to the defined type of the Named Variable. A new type can not be dynamically defined in DLMS.

10.5.2 Structure

Tableau 25 - Le service Write

Service Write	Req	Ind	Resp	Conf
Argument	M	M(=)		
List of Variable Access Specification	M	M(=)		
List of Data	M	M(=)		
Result(+)			S	S(=)
List of Write Result			M	M(=)
Success			S	S(=)
Data Access Error			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.5.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service Write.

Le paramètre List of Variable Access Specification (spécification d'une liste de variables à accéder) spécifie une ou plusieurs variables auxquelles il faut accéder. Le paramètre Variable Access Specification (spécification de l'accès aux variables) est décrit plus haut en 10.3.3.

Le paramètre List of Data (liste de données) contient les valeurs à écrire dans les variables spécifiées dans le paramètre List of Variable Access Specification. Les valeurs apparaissent dans la liste dans l'ordre des variables spécifiées par le paramètre List of Variable Access Specification. Le paramètre Data est décrit plus haut en 10.3.1.

Le paramètre Result(+) indique que le service demandé a fonctionné.

Le paramètre List of Write Result (liste des résultats d'écriture) contient les valeurs des variables spécifiées dans l'ordre spécifié par le paramètre List of Variable Access Specification de la primitive de demande. Chaque élément spécifie soit le succès de l'accès en écriture soit la raison pour laquelle l'écriture a échoué pour cette variable.

Le paramètre Success (succès), du type NULL, indique que l'écriture d'une variable donnée a réussi.

Le paramètre Data Access Error (erreur d'accès aux données), du type DataAccessError, fournit la raison de l'échec de l'écriture. La description de ce paramètre est détaillée en 10.3.2.

10.5.2 Structure

Table 25 - The Write service

Write service	Req	Ind	Resp	Conf
Argument	M	M(=)		
List of Variable Access Specification	M	M(=)		
List of Data	M	M(=)		
Result(+)			S	S(=)
List of Write Result			M	M(=)
Success			S	S(=)
Data Access Error			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.5.3 Parameters

The Argument parameter conveys the specific parameters of the Write service request.

The List of Variable Access Specification parameter specifies one or more variables which are to be accessed. The Variable Access Specification parameter is described above in 10.3.3.

The List of Data parameter contains the values to be written to the variables specified in the List of Variable Access Specification parameter. The values occur in the list in the order of the variables specified in the List of Variable Access Specification parameter. The Data parameter is described above in 10.3.1.

The Result(+) parameter indicates that the requested service has succeeded.

The List of Write Result parameter contains the values of the specified variables, in the order specified by the List of Variable Access Specification parameter of the request primitive. Each element specifies either the success of the write access or a reason for the write to fail for this variable.

The Success parameter, of type NULL, indicates that the write succeeded for a given variable.

The Data Access Error parameter, of type DataAccessError, provides the reason for the write to fail. The description of this parameter is detailed in 10.3.2.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit la raison de l'échec. Le paramètre Error Type est décrit en détail en A.5.

10.5.4 Procédures de service

Pour le service Write, un résultat positif signifie que la demande de service était acceptable pour le VDE et que le VDE a tenté de remplacer les valeurs de chacune des variables spécifiées par les valeurs fournies dans la demande.

Après avoir vérifié la validité de la demande de service, le VDE essaye d'écrire les valeurs des variable spécifiées. Pour chaque tentative il renvoie soit une confirmation du succès de l'écriture soit la raison de l'échec, dans l'ordre spécifié par le paramètre List of Variable Access Specification de la primitive de demande.

10.6 Le service UnconfirmedWrite *(écriture non confirmée)*

10.6.1 Objet

Le service UnconfirmedWrite a le même objet que le service Write mais il ne fournit pas de confirmation.

10.6.2 Structure

Tableau 26 - Le service UnconfirmedWrite

Service UnconfirmedWrite	Req	Ind
Argument	M	M(=)
List of Variable Access Specification	M	M(=)
List of Data	M	M(=)

Le service UnconfirmedWrite est un service qui ne donne pas de confirmation.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

10.5.4 Service procedure

For the Write service, a successful result means that the service request was acceptable to the VDE and that the VDE has attempted to replace the value of each of the specified variables with the values supplied in the request.

After verifying the validity of the service request, the VDE attempts to write the values of the specified variables. It returns for each write attempt, in the order specified in the List of Variable Access Specification parameter of the request primitive, a confirmation that the write succeeded or a reason for the failure.

10.6 UnconfirmedWrite service

10.6.1 Purpose

The UnconfirmedWrite service has the same purpose as the Write service but is an unconfirmed service.

10.6.2 Structure

Table 26 - The UnconfirmedWrite service

UnconfirmedWrite Service	Req	Ind
Argument	M	M(=)
List of Variable Access Specification	M	M(=)
List of Data	M	M(=)

The UnconfirmedWrite service is an unconfirmed service.

10.6.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service UnconfirmedWrite.

Le paramètre List of Variable Access Specification (spécification de la liste de variables à accéder) spécifie une ou plusieurs variables auxquelles on doit accéder. Le paramètre Variable Access Specification est décrit plus haut en 10.3.3.

Le paramètre List of Data (liste de données) contient les valeurs à écrire dans les variables spécifiées par le paramètre List of Variable Access Specification. Les valeurs apparaissent sur la liste dans l'ordre des variables spécifiées par le paramètre List of Variable Access Specification. Le paramètre Data est décrit plus haut en 10.3.1.

10.6.4 Procédures de service

Pour le service UnconfirmedWrite, un succès signifie que la demande de service était acceptable pour le VDE et que le VDE a essayé de remplacer les valeurs de chacune des variables par celles fournies dans la demande de service. Il n'y a pas de réponse renvoyée.

Après avoir vérifié la validité de la demande de service, le VDE tente d'écrire les valeurs des variables spécifiées.

10.7 Le service InformationReport *(notification)*

10.7.1 Objet

Le service InformationReport est un service spontané. Il est nécessaire à DLMS pour informer le client DLMS de la valeur d’une ou plusieurs variables spécifiées telles qu'elles auraient été lues par le client DLMS.

10.6.3 Parameters

The Argument parameter conveys the specific parameters of the UnconfirmedWrite service request.

The List of Variable Access Specification parameter specifies one or more variables which are to be accessed. The Variable Access Specification parameter is described above in 10.3.3.

The List of Data parameter contains the values to be written to the variables specified in the List of Variable Access Specification parameter. The values occur in the list in the order of the variables specified in the List of Variable Access Specification parameter. The Data parameter is described above in 10.3.1.

10.6.4 Service procedure

For the UnconfirmedWrite service, a success means that the service request was acceptable to the VDE and that the VDE has attempted to replace the value of each of the specified variables with the values supplied in the request. No response is returned.

After verifying the validity of the service request, the VDE attempts to write the values of the specified variables.

10.7. InformationReport service

10.7.1 Purpose

The InformationReport service is an unsolicited service. It is requested by the DLMS server in order to inform the DLMS client of the value of one or more specified variables, as though they had been read by the DLMS client.

10.7.2 Structure

Tableau 27 - Le service InformationReport

Service InformationReport	Req	Ind
Argument	M	M(=)
Current Time	M	M(=)
List of Variable Access Specification	M	M(=)
List of Data	M	M(=)

Le service InformationReport est un service sans confirmation qui est toujours initialisé par le serveur DLMS.

10.7.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service InformationReport.

Le paramètre Current Time (heure actuelle), du type GeneralizedTime (heure généralisée), indique l'heure à laquelle la primitive de service InformationReport a été émise.

Le paramètre List of Variable Access Specification (spécification de la liste de variables à accéder) spécifie les variables qui seront fournies au client. Ce paramètre est décrit plus haut en 10.3.3.

Le paramètre List of Data (liste de données) contient les valeurs des variables spécifiées, dans l'ordre défini par le paramètre List of Variable Access Specification précédent. Chaque élément de la liste est un paramètre Data. Il spécifie la valeur de la variable réelle au moment de l'accès. Le paramètre Data est décrit plus haut en 10.3.1.

10.7.2 Structure

Table 27 - The InformationReport service

InformationReport service	Req	Ind
Argument	M	M(=)
Current Time	M	M(=)
List of Variable Access Specification	M	M(=)
List of Data	M	M(=)

The InformationReport service is an unconfirmed service which is always initiated by the DLMS server.

10.7.3 Parameters

The Argument parameter conveys the specific parameters of the InformationReport service request.

The Current Time parameter, of type GeneralizedTime, indicates the time at which the InformationReport service primitive was issued.

The List Of Variable Access Specification parameter specifies the variables which will be provided to the client. The parameter is described above in 10.3.3.

The List of Data parameter contains the values of the specified variables, in the order specified by the previous List of Variable Access Specification parameter. Each element of the list is a Data parameter. It specifies the value of the real variable at the time of the access. The Data parameter is described above in 10.3.1.

10.7.4 Procédures de service

DLMS ne spécifie pas de procédure d'invocation et de réception du service InformationReport. C'est un problème local. Le VDE peut demander à tout moment ce service.

10.8 Le service GetVariableAttribute *(obtenir attributs des variables)*

10.8.1 Objet

Le service GetVariableAttribute sert à demander au serveur DLMS tous les attributs associés à un objet Variable.

10.8.2 Structure

Tableau 28 - Le service GetVariableAttribute

Service GetVariableAttribute	Req	Ind	Resp	Conf
Argument	M	M(=)		
Variable Name	M	M(=)		
Result(+)			S	S(=)
Named Variable			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
Type Description			M	M(=)
Read-Write Flag			M	M(=)
Available			M	M(=)
Named Variable List			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
List of Object Name			M	M(=)
Message Box			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
Read-Write Flag			M	M(=)
Available			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.7.4 Service procedure

DLMS does not specify a procedure for invoking and receiving the InformationReport service. It is used as a local action. At any time, the VDE may request this service.

10.8 GetVariableAttribute service

10.8.1 Purpose

The GetVariableAttribute service is used to request that a DLMS server returns all the attributes associated with a specified Variable object.

10.8.2 Structure

Table 28 - The GetVariableAttribute service

GetVariableAttribute service	Req	Ind	Resp	Conf
Argument	M	M(=)		
Variable Name	M	M(=)		
Result(+)			S	S(=)
Named Variable			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
Type Description			M	M(=)
Read-Write Flag			M	M(=)
Available			M	M(=)
Named Variable List			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
List of Object Name			M	M(=)
Message Box			S	S(=)
Scope Of Access			M	M(=)
Scope May Change			M	M(=)
Lifetime			M	M(=)
Read-Write Flag			M	M(=)
Available			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

10.8.3 Paramètres

Le paramètre Argument véhicule les paramètres spécifiques de la demande de service GetVariableAttribute.

Le paramètre Variable Name (nom de variable), du type ObjectName, identifie de manière unique dans le VDE l'objet variable concerné. Le nom de variable peut être le nom d'un objet Variable Désignée, le nom d'un objet Boîte à Messages ou d'un objet Liste de Variables Désignées.

Le paramètre Result(+), du type CHOICE (choix), indique que le service demandé a fonctionné. Il identifie la classe de l'objet recherché. Les valeurs possibles de ce paramètres sont : NAMED-VARIABLE (variable désignée), NAMED-VARIABLE-LIST (liste de variables désignées) ou MESSAGE-BOX (boîte à messages).

Le paramètre Named Variable (variable désignée) spécifie que la classe de l'objet recherché est NAMED-VARIABLE.

Le paramètre Scope Of Access (champ d'accès), du type ScopeOfAccess, précise le champ d'accès actuel d'accès de l'objet désigné.

Le paramètre Scope May Change (champ peut changer), du type BOOLEAN (booléen) définit si oui (TRUE) ou non (FALSE) le champ d'accès de l'objet concerné peut être modifié à l'aide du service ChangeScope de DLMS.

Le paramètre Lifetime (durée de vie), de type BOOLEAN (booléen), spécifie si la durée de vie est VDE (TRUE) (vrai) ou DATA-SET (FALSE) (jeu de données - faux). L'attribut Lifetime positionné sur DATA-SET indique que cet objet est défini dans le jeu de données.

Le paramètre Type Description (description de type), de type TypeDescription, indique pour les besoins de l'accès, le type abstrait de la variable réelle sous-jacente telle qu'elle est vue en utilisant les services de DLMS. Il doit spécifier la classe (nombre entier, nombre à virgule flottante, ..) et la plage des valeurs possibles des éléments simples de la variable, ainsi que, lorsque c'est applicable, le regroupement de ces éléments de données (dans des tableaux ou des structures).

Le paramètre Read-Write Flag (drapeau lecture-écriture), du type BOOLEAN (booléen), indique si le droit de lecture et d'écriture de l'objet est : READ-WRITE (TRUE) (lecture et écriture - vrai) ou READ-ONLY (FALSE) (lecture seulement - faux). Cette valeur peut dépendre du VAA.

Le paramètre Available (disponible), de type BOOLEAN (booléen), spécifie si oui (TRUE) ou non (FALSE) la variable réelle sous-jacente est actuellement disponible quand on considère l'accès physique et les ressources matérielles. L'attribut Available et sa gestion dépendent de l'implémentation.

10.8.3 Parameters

The Argument parameter conveys the specific parameter of the GetVariableAttribute service request.

The Variable Name parameter, of type ObjectName, identifies uniquely within a VDE the involved variable object. The variable name may be a Named Variable object name, a Message Box object name or a Named Variable List object name.

The Result(+) parameter, of type CHOICE, indicates that the requested service has succeeded. It identifies the class of the requested object. The possible values for this parameter are: NAMED-VARIABLE, NAMED-VARIABLE-LIST or MESSAGE-BOX.

The Named Variable parameter specifies that the class of the requested object is NAMED-VARIABLE.

The Scope Of Access parameter, of type ScopeOfAccess, specifies the current scope of access for the designated object.

The Scope May Change parameter, of type BOOLEAN, defines whether (TRUE) or not (FALSE) the scope of access of the related object may change using the DLMS ChangeScope service.

The Lifetime parameter, of type BOOLEAN, specifies whether the described object lifetime is VDE (TRUE) or DATA-SET (FALSE). A Lifetime attribute set to DATA-SET indicates that the object is defined within the Data Set.

The Type Description parameter, of type TypeDescription, indicates the abstract type of the underlying real variable, when viewed using DLMS services. It shall specify the class (integer, floating-point, ...) and range of values for the simple data elements of the variable, as well as the grouping of these data elements (into arrays or structures), if applicable, for the purpose of access.

The Read-Write Flag parameter, of type BOOLEAN, indicates whether the object read and write right is READ-WRITE (TRUE) or READ-ONLY (FALSE). The value of the parameter may depend on the VAA.

The Available parameter, of type BOOLEAN, specifies whether (TRUE) or not (FALSE) the underlying real variable is currently available considering physical access and hardware resources. The Available attribute and its management are implementation dependent.

Le paramètre Named Variable List (liste de variables désignées), spécifique la classe de l'objet demandé est NAMED-VARIABLE-LIST.

Le paramètre List of Object Name (liste de noms d'objets), du type SEQUENCE OF ObjectName (séquence de noms d'objets), contient la liste des noms des objets Variables Désignées qui constitue l'objet Liste de Variables Désignées.

Les autres paramètres du paramètre Named Variable List sont identiques à ceux décrits pour le paramètre Named Variable. Se reporter ci-dessus pour leur description.

Le paramètre Message Box (boîte à messages) spécifie que la classe de l'objet demandé est MESSAGE-BOX.

Les paramètres du paramètre Message Box sont identiques à ceux décrits pour le paramètre Named Variable. Se référer à leur description ci-dessus.

Le paramètre Result(-) indique que le service demandé auparavant a échoué. Le paramètre Error Type fournit les raisons de l'échec. Le paramètre Error Type est décrit de façon détaillée en A.5.

10.8.4 Procédure de service

Le serveur DLMS renvoie les attributs associés à l'objet variable spécifié. Les restrictions d'accès VAA (association virtuelle d'applications) s'appliquent lors de l'utilisation de ce service.

The Named Variable List parameter specifies that the class of the requested object is NAMED-VARIABLE-LIST.

The List of Object Name parameter, of type SEQUENCE OF ObjectName, contains the list of Named Variable object names that constitute the Named Variable List object.

The other parameters of the Named Variable List parameter are identical to those described for the Named Variable parameter. Please refer above for their description.

The Message Box parameter specifies that the class of the requested object is MESSAGE-BOX.

The parameters of the Message Box parameter are identical to those described for the Named Variable parameter. Please refer above for their description.

The Result(-) parameter indicates that the service previously requested failed. The Error Type parameter provides the reason for failure. The Error Type parameter is described in detail in A.5.

10.8.4 Service procedure

The DLMS server returns the attributes associated to the specified variable object. The VAA access restrictions apply to the use of this service.

Annexe A
(normative)

Protocole DLMS

L'objet de la présente annexe est de décrire le protocole DLMS à l'aide des DLMS ASE (Service d'Eléments d'Association) qui sont décrits dans la CEI/FDIS 1334-4-42.

NOTE – Cette annexe a été attachée à la présente norme plutôt qu'à la future CEI 1334-4-42, car les descriptions de types utilisent beaucoup de paramètres de DLMS. Elle doit être lue après les spécifications de services détaillées dans les articles ci-dessus.

A.1 Conventions

Cette annexe utilise les conventions de description contenues dans les Conventions de Service OSI (ISO/CEI/TR 8509). Le modèle définit les interactions entre le client DLMS et le serveur DLMS. L'information est échangée entre les utilisateurs DLMS à l'aide des primitives de services, qui peuvent véhiculer des paramètres.

A.1.1 Notation

Cette annexe utilise la notation de syntaxe abstraite définie dans l'ISO/CEI 8824 (Spécification de ASN.1). Pour respecter les intentions et les exigences de la norme ASN.1, tous les symboles non terminaux (par exemple ceux qui apparaissent à gauche d'une production) commencent par une lettre majuscule. Tous les identificateurs, valeurs de référence et types de référence commencent avec une lettre minuscule.

Sauf information contraire, la base dix est utilisée pour représenter les valeurs numériques.

A.1.2 Paramètres d'évitement

Beaucoup des paramètres des différents services de DLMS sont simplement transmis d'une primitive request (demande) à une primitive indication via le service request PDU ou d'une primitive response (réponse) à une primitive confirm (confirmation) via le service response PDU, sans qu'aucune autre action ne soit entreprise par le prestataire DLMS.

Si un paramètre est optionnel et s'il est absent dans la primitive request du service, il doit aussi être absent dans le request PDU. Si un paramètre optionnel est absent dans le request PDU, il doit aussi être absent dans la primitive indication du service.

Annex A
(normative)

DLMS protocol

The purpose of this annex is to describe the DLMS protocol using the DLMS ASE that are described in IEC/FDIS 1334-4-42.

NOTE – This annex is placed in this standard rather than in future IEC 1334-4-42 because the type descriptions use many of the DLMS services parameters. It must be read after the service specifications detailed in the clauses above.

A.1 Conventions

This annex uses the descriptive conventions contained in the OSI Service Conventions (ISO/IEC/TR 8509). The model defines the interactions between the DLMS client and the DLMS server. Information is passed between the DLMS users by service primitives, which may convey parameters.

A.1.1 Notation

This annex uses the abstract syntax notation defined in ISO/IEC 8824 (ASN.1 Specification). In keeping with the intent and requirements of the ASN.1 Standard, all non-terminal symbols (that is those which appear on the left-hand side of a production) begin with a capital letter. All identifiers, value references and type references begin with a lower-case letter.

Unless otherwise noted, a base ten representation for all numeric values is used.

A.1.2 Pass-through parameters

Many of the parameters of the various DLMS services are simply passed from the request primitive via the service request PDU to the indication primitive or from the response primitive via the service response PDU to the confirm primitive, without other action being taken by the DLMS provider.

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

Si un paramètre est optionnel et s'il est omis dans la primitive response (réponse) du service, il doit être absent dans la response PDU. Si un paramètre optionnel est absent de la response PDU, il doit être absent dans la primitive confirm (confirmation) du service.

A.1.3 Valeurs énumérées dans les paramètres

Pour les paramètres qui ont des valeurs énumérées, la valeur spécifiée pour le paramètre du protocole correspondant doit être la valeur de même nom pour la primitive du service contenant le paramètre. Les valeurs véhiculées dans la primitive du service, la PDU qui en résulte et la primitive du service qui résulte de la réception de cette primitive du service doivent être sémantiquement équivalents, comme il est spécifié dans ASN.1.

NOTE – La correspondance entre de telles valeurs est identifiée dans cette spécification par l'utilisation des mêmes noms dans les primitives des services et les protocoles. Dans les spécifications de services, de telles valeurs sont spécifiées en caractères majuscules. Dans les spécifications de protocoles, le cas d'usage du nom est choisi de façon à satisfaire les exigences de la syntaxe ASN.1.

A.1.4 Confirmation négative

La plupart des services confirmés de DLMS incluent une confirmation négative dans le cas où il survient une erreur lors du traitement de la primitive request (demande) par le serveur DLMS. Cette confirmation négative est notifiée par le paramètre Result(-) et un paramètre Error Type dans la primitive response (réponse) du service. Un paramètre Result(-) et un paramètre Error Type sémantiquement équivalents à ces paramètres dans la primitive response (réponse) doivent apparaître dans la primitive confirm (confirmation) du service.

La syntaxe abstraite pour une confirmation négative doit être ServiceError DLMSpdu avec le champ error (erreur) dérivé du paramètre problem (problème) dans la primitive response du service.

A.1.5 Définitions ASN.1

Les définitions ASN.1 fournies dans cette annexe font partie du Module ASN.1 "DCP-DLMS". Les déclarations de début et de fin, indiquant que chaque définition ASN.1 fournie fait partie de ce module, sont omises afin rendre plus facile la lecture du document.

Ainsi, chaque définition contient implicitement au début la déclaration :

```
DCP-DLMS DEFINITIONS ::= BEGIN
```

et à la fin la déclaration :

```
END.
```

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

A.1.3 Enumerated values in parameters

For the parameters in the service description that have enumerated values, the value specified for the corresponding protocol parameter shall be the value of the same name from the service primitive containing the parameter. The values conveyed in the service primitive, resulting PDU, and the service primitive that results from receipt of the service primitive shall be semantically equivalent as specified in ASN.1.

NOTE – The correspondence between such values is identified in this specification through the use of the same names in the service primitives and protocol. In the service specification, such values are specified in all upper-case characters. In the protocol specification, the case of the name is chosen so as to satisfy ASN.1 syntax requirements.

A.1.4 Negative confirmation

Most confirmed DLMS services include a negative confirmation in the case that an error occurs in the processing of the service request by the DLMS server. Such negative confirmation shall be indicated by a Result(-) parameter and an Error Type parameter in the service response primitive. A Result(-) parameter and an Error Type parameter which are semantically equivalent to those parameters in the response primitive shall appear in the confirm service primitive.

The abstract syntax for a negative confirmation shall be a ServiceError DLMSpdu with the error field derived from the problem parameter in the response service primitive.

A.1.5 ASN.1 definitions

The ASN.1 definitions provided in this annex are part of the ASN.1 Module "DCP-DLMS". The beginning and closing statements, which indicate that the ASN.1 definition provided is part of this module, are omitted in order to make the reading of this document easier.

Therefore, each ASN.1 definition implicitly contains at the beginning the statement:

DCP-DLMS DEFINITIONS ::= BEGIN

and at the end the statement:

END.

A.2 Types utiles

Quelques types utiles sont décrits dans ce paragraphe. Ces types sont finalement utilisés pour les descriptions de type du protocole DLMS.

Integer8	::= INTEGER(-128..127)	- - Integer on 8 bits
Integer16	::= INTEGER(-32768..32767)	- - Integer on 16 bits
Integer32	::= INTEGER(-2 147 483 648..2 147 483 647)	- - Integer on 32 bits
Unsigned8	::= INTEGER(0..127)	- - Unsigned on 8 bits
Unsigned16	::= INTEGER(0..32767)	- - Unsigned on 16 bits
Unsigned32	::= INTEGER(0.. 2 147 483 647)	- - Unsigned on 32 bits

Les types OCTET STRING[1)] (chaîne d'octets), BIT STRING[1)] (chaîne binaire), VisibleString[1)] (chaîne en clair) et GeneralizedTime[1)] (temps généralisé) sont définis dans ASN.1 dans l'article "Useful Types" (types utiles).

1) *Attention : A cause de contraintes physiques, la longueur de ces chaînes peut être limitées dans certains profils. Par exemple, dans les systèmes à courants porteurs sur ligne de distribution, ces chaînes sont limitées à 230 octets.*

A.2 Useful types

Some useful types are described in this subclause. These types are finally used by the type descriptions of the DLMS protocol.

Integer8	::= INTEGER(-128..127)	-- Integer on 8 bits
Integer16	::= INTEGER(-32768..32767)	-- Integer on 16 bits
Integer32	::= INTEGER(-2 147 483 648..2 147 483 647)	-- Integer on 32 bits
Unsigned8	::= INTEGER(0..127)	-- Unsigned on 8 bits
Unsigned16	::= INTEGER(0..32767)	-- Unsigned on 16 bits
Unsigned32	::= INTEGER(0..2 147 483 647)	-- Unsigned on 32 bits

The OCTET STRING[1)] , BIT STRING[1)], VisibleString[1)] and GeneralizedTime[1)] types are defined within ASN.1 in the Useful Types clause.

1) *Warning: Because of physical constraints, the length of these strings may be limited within some profiles. For example, in power line carrier systems on distribution lines, these strings are limited to 230 octets.*

A.3 DLMS PDU

Les PDU utilisées pour gérer le protocole DLMS sont décrites comme suit :

```
DLMSpdu::= CHOICE {
    -- DLMS PDUs (no encryption selected)
    confirmedServiceRequest              [0]              ConfirmedServiceRequest,
    initiateRequest                      [1] IMPLICIT     InitiateRequest,
    getStatusRequest                     [2] IMPLICIT     GetStatusRequest,
    getNameListRequest                   [3] IMPLICIT     GetNameListRequest,
    getVariableAttributeRequest          [4] IMPLICIT     GetVariableAttributeRequest,
    readRequest                          [5] IMPLICIT     ReadRequest,
    writeRequest                         [6] IMPLICIT     WriteRequest,

    confirmedServiceResponse             [7]              ConfirmedServiceResponse,
    initiateResponse                     [8] IMPLICIT     InitiateResponse,
    getStatusResponse                    [9] IMPLICIT     GetStatusResponse,
    getNameListResponse                  [10] IMPLICIT    GetNameListResponse,
    getVariableAttributeResponse         [11]             GetVariableAttributeResponse,
    readResponse                         [12] IMPLICIT    ReadResponse,
    writeResponse                        [13] IMPLICIT    WriteResponse,

    confirmedServiceError                [14]             ConfirmedServiceError,

    unconfirmedServiceRequest            [20]             UnconfirmedServiceRequest
    abortRequest                         [21] IMPLICIT    AbortRequest,
    unconfirmedWriteRequest              [22] IMPLICIT    UnconfirmedWriteRequest,

    unsolicitedServiceRequest            [23]             UnsolicitedServiceRequest,
    informationReportRequest             [24] IMPLICIT    InformationReportRequest,

    -- The following choices may only be selected when using ciphered DLMS.

    -- DASE PDUs (global-ciphering)
    glo-confirmedServiceRequest          [32] IMPLICIT OCTET STRING,
    glo-initiateRequest                  [33] IMPLICIT OCTET STRING,
    glo-getStatusRequest                 [34] IMPLICIT OCTET STRING,
    glo-getNameListRequest               [35] IMPLICIT OCTET STRING,
    glo-getVariableAttributeRequest      [36] IMPLICIT OCTET STRING,
    glo-readRequest                      [37] IMPLICIT OCTET STRING,
    glo-writeRequest                     [38] IMPLICIT OCTET STRING,

    glo-confirmedServiceResponse         [39] IMPLICIT OCTET STRING,
    glo-initiateResponse                 [40] IMPLICIT OCTET STRING,
    glo-getStatusResponse                [41] IMPLICIT OCTET STRING,
    glo-getNameListResponse              [42] IMPLICIT OCTET STRING,
    glo-getVariableAttributeResponse     [43] IMPLICIT OCTET STRING,
    glo-readResponse                     [44] IMPLICIT OCTET STRING,
    glo-writeResponse                    [45] IMPLICIT OCTET STRING,

    glo-confirmedServiceError            [46] IMPLICIT OCTET STRING,

    glo-unconfirmedServiceRequest        [52] IMPLICIT OCTET STRING,
    glo-abortRequest                     [53] IMPLICIT OCTET STRING,
    glo-unconfirmedWriteRequest          [54] IMPLICIT OCTET STRING,

    glo-unsolicitedServiceRequest        [55] IMPLICIT OCTET STRING,
    glo-informationReportRequest         [56] IMPLICIT OCTET STRING,
```

A.3 DLMS PDU

The PDUs used to operate the DLMS protocol are described as follows:

```
DLMSpdu::= CHOICE {
    -- DLMS PDUs (no encryption selected)
    confirmedServiceRequest              [0]              ConfirmedServiceRequest,
    initiateRequest                      [1] IMPLICIT     InitiateRequest,
    getStatusRequest                     [2] IMPLICIT     GetStatusRequest,
    getNameListRequest                   [3] IMPLICIT     GetNameListRequest,
    getVariableAttributeRequest          [4] IMPLICIT     GetVariableAttributeRequest,
    readRequest                          [5] IMPLICIT     ReadRequest,
    writeRequest                         [6] IMPLICIT     WriteRequest,

    confirmedServiceResponse             [7]              ConfirmedServiceResponse,
    initiateResponse                     [8] IMPLICIT     InitiateResponse,
    getStatusResponse                    [9] IMPLICIT     GetStatusResponse,
    getNameListResponse                  [10] IMPLICIT    GetNameListResponse,
    getVariableAttributeResponse         [11]             GetVariableAttributeResponse,
    readResponse                         [12] IMPLICIT    ReadResponse,
    writeResponse                        [13] IMPLICIT    WriteResponse,

    confirmedServiceError                [14]             ConfirmedServiceError,

    unconfirmedServiceRequest            [20]             UnconfirmedServiceRequest
    abortRequest                         [21] IMPLICIT    AbortRequest,
    unconfirmedWriteRequest              [22] IMPLICIT    UnconfirmedWriteRequest,

    unsolicitedServiceRequest            [23]             UnsolicitedServiceRequest,
    informationReportRequest             [24] IMPLICIT    InformationReportRequest,

    -- The following choices may only be selected when using ciphered DLMS.

    -- DASE PDUs (global-ciphering)
    glo-confirmedServiceRequest          [32] IMPLICIT OCTET STRING,
    glo-initiateRequest                  [33] IMPLICIT OCTET STRING,
    glo-getStatusRequest                 [34] IMPLICIT OCTET STRING,
    glo-getNameListRequest               [35] IMPLICIT OCTET STRING,
    glo-getVariableAttributeRequest      [36] IMPLICIT OCTET STRING,
    glo-readRequest                      [37] IMPLICIT OCTET STRING,
    glo-writeRequest                     [38] IMPLICIT OCTET STRING,

    glo-confirmedServiceResponse         [39] IMPLICIT OCTET STRING,
    glo-initiateResponse                 [40] IMPLICIT OCTET STRING,
    glo-getStatusResponse                [41] IMPLICIT OCTET STRING,
    glo-getNameListResponse              [42] IMPLICIT OCTET STRING,
    glo-getVariableAttributeResponse     [43] IMPLICIT OCTET STRING,
    glo-readResponse                     [44] IMPLICIT OCTET STRING,
    glo-writeResponse                    [45] IMPLICIT OCTET STRING,

    glo-confirmedServiceError            [46] IMPLICIT OCTET STRING,

    glo-unconfirmedServiceRequest        [52] IMPLICIT OCTET STRING,
    glo-abortRequest                     [53] IMPLICIT OCTET STRING,
    glo-unconfirmedWriteRequest          [54] IMPLICIT OCTET STRING,

    glo-unsolicitedServiceRequest        [55] IMPLICIT OCTET STRING,
    glo-informationReportRequest         [56] IMPLICIT OCTET STRING,
```

```
-- DASE PDUs (dedicated-ciphering)
ded-confirmedServiceRequest          [64] IMPLICIT OCTET STRING,
ded-initiateRequest                  [65] IMPLICIT OCTET STRING,
ded-getStatusRequest                 [66] IMPLICIT OCTET STRING,
ded-getNameListRequest               [67] IMPLICIT OCTET STRING,
ded-getVariableAttributeRequest      [68] IMPLICIT OCTET STRING,
ded-readRequest                      [69] IMPLICIT OCTET STRING,
ded-writeRequest                     [70] IMPLICIT OCTET STRING,

ded-confirmedServiceResponse         [71] IMPLICIT OCTET STRING,
ded-initiateResponse                 [72] IMPLICIT OCTET STRING,
ded-getStatusResponse                [73] IMPLICIT OCTET STRING,
ded-getNameListResponse              [74] IMPLICIT OCTET STRING,
ded-getVariableAttributeResponse     [75] IMPLICIT OCTET STRING,
ded-readResponse                     [76] IMPLICIT OCTET STRING,
ded-writeResponse                    [77] IMPLICIT OCTET STRING,

ded-confirmedServiceError            [78] IMPLICIT OCTET STRING,

ded-unconfirmedServiceRequest        [84] IMPLICIT OCTET STRING,
ded-abortRequest                     [85] IMPLICIT OCTET STRING,
ded-unconfirmedWriteRequest          [86] IMPLICIT OCTET STRING,

ded-unsolicitedServiceRequest        [87] IMPLICIT OCTET STRING,
ded-informationReportRequest         [88] IMPLICIT OCTET STRING
}
```

Les OCTET STRING qui apparaissent dans les PDU chiffrées sont construites par application d'un algorithme de chiffrement aux PDU correspondantes non chiffrées. Cet algorithme peut inclure, selon le choix de l'émetteur, l'insertion d'un champ "copy-check" qui identifie de façon unique la PDU émise, afin de permettre la détection de retransmissions par des personnes non autorisées d'une PDU chiffrée précédemment émise. L'algorithme de chiffrement peut inclure aussi l'incorporation d'un checksum afin de vérifier qu'une PDU valide existe dans le message déchifffré.

Chaque DLMS PDU contient un identificateur du type de PDU.

```
-- DASE PDUs (dedicated-ciphering)
ded-confirmedServiceRequest          [64] IMPLICIT OCTET STRING,
ded-initiateRequest                  [65] IMPLICIT OCTET STRING,
ded-getStatusRequest                 [66] IMPLICIT OCTET STRING,
ded-getNameListRequest               [67] IMPLICIT OCTET STRING,
ded-getVariableAttributeRequest      [68] IMPLICIT OCTET STRING,
ded-readRequest                      [69] IMPLICIT OCTET STRING,
ded-writeRequest                     [70] IMPLICIT OCTET STRING,

ded-confirmedServiceResponse         [71] IMPLICIT OCTET STRING,
ded-initiateResponse                 [72] IMPLICIT OCTET STRING,
ded-getStatusResponse                [73] IMPLICIT OCTET STRING,
ded-getNameListResponse              [74] IMPLICIT OCTET STRING,
ded-getVariableAttributeResponse     [75] IMPLICIT OCTET STRING,
ded-readResponse                     [76] IMPLICIT OCTET STRING,
ded-writeResponse                    [77] IMPLICIT OCTET STRING,

ded-confirmedServiceError            [78] IMPLICIT OCTET STRING,

ded-unconfirmedServiceRequest        [84] IMPLICIT OCTET STRING,
ded-abortRequest                     [85] IMPLICIT OCTET STRING,
ded-unconfirmedWriteRequest          [86] IMPLICIT OCTET STRING,

ded-unsolicitedServiceRequest        [87] IMPLICIT OCTET STRING,
ded-informationReportRequest         [88] IMPLICIT OCTET STRING
}
```

The OCTET STRINGs that appear in the ciphered PDUs are constructed by applying a ciphering algorithm to the corresponding non-ciphered PDU. This ciphering algorithm may include, at the discretion of the sending user, the insertion of a 'copy-check' field that uniquely identifies the instance of PDU being sent, so as to detect any retransmissions by a non-authorized user of a previously sent ciphered PDU. The ciphering algorithm may include also the incorporation of a checksum to verify that a valid PDU exists in the de-ciphered message.

Each DLMS PDU contains an identifier of the PDU type.

A.4 Service request and response (demandes et réponses)

```
ConfirmedServiceRequest::= CHOICE {
        -- tags 0 to 6 are reserved
        getDataSetAttribute     [7] IMPLICIT    GetDataSetAttributeRequest,
        getTIAttribute          [8] IMPLICIT    GetTIAttributeRequest,
        changeScope             [9] IMPLICIT    ChangeScopeRequest,
        start                   [10] IMPLICIT   StartRequest,
        stop                    [11] IMPLICIT   StopRequest,
        resume                  [12] IMPLICIT   ResumeRequest,
        makeUsable              [13] IMPLICIT   MakeUsableRequest,
        initiateLoad            [14] IMPLICIT   InitiateLoadRequest,
        loadSegment             [15] IMPLICIT   LoadSegmentRequest,
        terminateLoad           [16] IMPLICIT   TerminateLoadRequest
        initiateUpLoad          [17] IMPLICIT   InitiateUpLoadRequest,
        upLoadSegment           [18] IMPLICIT   UpLoadSegmentRequest,
        terminateUpLoad         [19] IMPLICIT   TerminateUpLoadRequest
        }

ConfirmedServiceResponse::= CHOICE {
        -- tags 0 to 13 are reserved
        getDataSetAttribute     [14] IMPLICIT   GetDataSetAttributeResponse,
        getTIAttribute          [15] IMPLICIT   GetTIAttributeResponse,
        changeScope             [16] IMPLICIT   ChangeScopeResponse,
        start                   [17] IMPLICIT   StartResponse,
        stop                    [18] IMPLICIT   StopResponse,
        resume                  [19] IMPLICIT   ResumeResponse,
        makeUsable              [20] IMPLICIT   MakeUsableResponse,
        initiateLoad            [21] IMPLICIT   InitiateLoadResponse,
        loadSegment             [22] IMPLICIT   LoadSegmentResponse,
        terminateLoad           [23] IMPLICIT   TerminateLoadResponse
        initiateUpLoad          [24] IMPLICIT   InitiateUpLoadResponse,
        upLoadSegment           [25] IMPLICIT   UpLoadSegmentResponse,
        terminateUpLoad         [26] IMPLICIT   TerminateUpLoadResponse
        }
```

A.4 Service requests and responses

```
ConfirmedServiceRequest::= CHOICE {
        -- tags 0 to 6 are reserved
        getDataSetAttribute     [7] IMPLICIT    GetDataSetAttributeRequest,
        getTIAttribute          [8] IMPLICIT    GetTIAttributeRequest,
        changeScope             [9] IMPLICIT    ChangeScopeRequest,
        start                   [10] IMPLICIT   StartRequest,
        stop                    [11] IMPLICIT   StopRequest,
        resume                  [12] IMPLICIT   ResumeRequest,
        makeUsable              [13] IMPLICIT   MakeUsableRequest,
        initiateLoad            [14] IMPLICIT   InitiateLoadRequest,
        loadSegment             [15] IMPLICIT   LoadSegmentRequest,
        terminateLoad           [16] IMPLICIT   TerminateLoadRequest
        initiateUpLoad          [17] IMPLICIT   InitiateUpLoadRequest,
        upLoadSegment           [18] IMPLICIT   UpLoadSegmentRequest,
        terminateUpLoad         [19] IMPLICIT   TerminateUpLoadRequest
        }

ConfirmedServiceResponse::= CHOICE {
        -- tags 0 to 13 are reserved
        getDataSetAttribute     [14] IMPLICIT   GetDataSetAttributeResponse,
        getTIAttribute          [15] IMPLICIT   GetTIAttributeResponse,
        changeScope             [16] IMPLICIT   ChangeScopeResponse,
        start                   [17] IMPLICIT   StartResponse,
        stop                    [18] IMPLICIT   StopResponse,
        resume                  [19] IMPLICIT   ResumeResponse,
        makeUsable              [20] IMPLICIT   MakeUsableResponse,
        initiateLoad            [21] IMPLICIT   InitiateLoadResponse,
        loadSegment             [22] IMPLICIT   LoadSegmentResponse,
        terminateLoad           [23] IMPLICIT   TerminateLoadResponse
        initiateUpLoad          [24] IMPLICIT   InitiateUpLoadResponse,
        upLoadSegment           [25] IMPLICIT   UpLoadSegmentResponse,
        terminateUpLoad         [26] IMPLICIT   TerminateUpLoadResponse
        }
```

```
ConfirmedServiceError::= CHOICE {
        -- tag 0 is reserved
        initiateError               [1]         ServiceError,
        getStatus                   [2]         ServiceError,
        getNameList                 [3]         ServiceError,
        getVariableAttribute        [4]         ServiceError,
        read                        [5]         ServiceError,
        write                       [6]         ServiceError,
        getDataSetAttribute         [7]         ServiceError,
        getTIAttribute              [8]         ServiceError,
        changeScope                 [9]         ServiceError,
        start                       [10]        ServiceError,
        stop                        [11]        ServiceError,
        resume                      [12]        ServiceError,
        makeUsable                  [13]        ServiceError,
        initiateLoad                [14]        ServiceError,
        loadSegment                 [15]        ServiceError,
        terminateLoad               [16]        ServiceError
        initiateUpLoad              [17]        ServiceError,
        upLoadSegment               [18]        ServiceError,
        terminateUpLoad             [19]        ServiceError
        }


UnconfirmedServiceRequest::= NULL

        -- des extensions futures sont à l'étude

UnsolicitedServiceRequest::= NULL

        -- des extensions futures sont à l'étude
```

```
ConfirmedServiceError::= CHOICE {
        -- tag 0 is reserved
        initiateError               [1]     ServiceError,
        getStatus                   [2]     ServiceError,
        getNameList                 [3]     ServiceError,
        getVariableAttribute        [4]     ServiceError,
        read                        [5]     ServiceError,
        write                       [6]     ServiceError,
        getDataSetAttribute         [7]     ServiceError,
        getTIAttribute              [8]     ServiceError,
        changeScope                 [9]     ServiceError,
        start                       [10]    ServiceError,
        stop                        [11]    ServiceError,
        resume                      [12]    ServiceError,
        makeUsable                  [13]    ServiceError,
        initiateLoad                [14]    ServiceError,
        loadSegment                 [15]    ServiceError,
        terminateLoad               [16]    ServiceError
        initiateUpLoad              [17]    ServiceError,
        upLoadSegment               [18]    ServiceError,
        terminateUpLoad             [19]    ServiceError
        }


UnconfirmedServiceRequest::= NULL

        -- further extensions are under consideration

UnsolicitedServiceRequest::= NULL

        -- further extensions are under consideration
```

A.5 Service error (erreur)

```
ServiceError::= CHOICE {
    application-reference      [0]          IMPLICIT  ENUMERATED {
        -- DLMS Provider only
        other                               (0),
        time-elapsed                        (1),      -- time out since request sent
        application-unreachable             (2),      -- peer AEi not reacheable
        application-reference-invalid       (3),      -- addressing trouble
        application-context-unsupported     (4),      -- application-context incompatibility
        provider-communication-error        (5),      -- error at the local or distant equipment
        deciphering-error                   (6)       -- error detected by the deciphering function
        },
    hardware-resource          [1]          IMPLICIT  ENUMERATED {
        -- VDE hardware troubles
        other                               (0),
        memory-unavailable                  (1),
        processor-resource-unavailable      (2),
        mass-storage-unavailable            (3),
        other-resource-unavailable          (4)
        },
    vde-state-error            [2]          IMPLICIT  ENUMERATED {
        -- Error source description
        other                               (0),
        no-dlms-context                     (1),
        loading-data-set                    (2),
        status-nochange                     (3),
        status-inoperable                   (4)
        },
    service                    [3]          IMPLICIT  ENUMERATED {
        -- service handling troubles
        other                               (0),
        pdu-size                            (1), -- pdu too long (refer to Companion Specification)
        service-unsupported                 (2)  -- as described in the Conformance Block
        },
    definition                 [4]          IMPLICIT  ENUMERATED {
        -- object bound troubles in a service
        other                               (0),
        object-undefined                    (1), -- object not defined at the VDE
        object-class-inconsistent           (2), -- class of object incompatible with asked service
        object-attribute-inconsistent       (3)  -- object attributes are inconsistent
        },
    access                     [5]          IMPLICIT  ENUMERATED {
        -- object access error
        other                               (0),
        scope-of-access-violated            (1), -- access denied through authorization reason
        object-access-invalid               (2), -- access incompatible with object attribute
        hardware-fault                      (3), -- access fail for hardware reason
        object-unavailable                  (4)  -- VDE hands object for unavailable
        },
```

A.5 Service error

```
ServiceError::= CHOICE {
    application-reference     [0]          IMPLICIT ENUMERATED {
        -- DLMS Provider only
        other                               (0),
        time-elapsed                        (1),  -- time out since request sent
        application-unreachable             (2),  -- peer AEi not reacheable
        application-reference-invalid       (3),  -- addressing trouble
        application-context-unsupported     (4),  -- application-context incompatibility
        provider-communication-error        (5),  -- error at the local or distant equipment
        deciphering-error                   (6)   -- error detected by the deciphering function
        },
    hardware-resource         [1]          IMPLICIT        ENUMERATED {
        -- VDE hardware troubles
        other                               (0),
        memory-unavailable                  (1),
        processor-resource-unavailable      (2),
        mass-storage-unavailable            (3),
        other-resource-unavailable          (4)
        },
    vde-state-error           [2]          IMPLICIT        ENUMERATED {
         -- Error source description
        other                               (0),
        no-dlms-context                     (1),
        loading-data-set                    (2),
        status-nochange                     (3),
        status-inoperable                   (4)
        },
    service                   [3]          IMPLICIT        ENUMERATED {
        -- service handling troubles
        other                               (0),
        pdu-size                            (1), -- pdu too long (refer to Companion Specification)
        service-unsupported                 (2)  -- as described in the Conformance Block
        },
    definition                [4]          IMPLICIT        ENUMERATED {
         -- object bound troubles in a service
        other                               (0),
        object-undefined                    (1), -- object not defined at the VDE
        object-class-inconsistent           (2), -- class of object incompatible with asked service
        object-attribute-inconsistent       (3)  -- object attributes are inconsistent
        },
    access                    [5]          IMPLICIT        ENUMERATED {
         -- object access error
        other                               (0),
        scope-of-access-violated            (1), -- access denied through authorization reason
        object-access-invalid               (2), -- access incompatible with object attribute
        hardware-fault                      (3), -- access fail for hardware reason
        object-unavailable                  (4)  -- VDE hands object for unavailable
        },
```

```
initiate                    [6]     IMPLICIT  ENUMERATED {
        -- initiate service error
        other                       (0),
        dlms-version-too-low        (1), -- proposed DLMS version too low
        incompatible-conformance    (2), -- proposed services not sufficient
        pdu-size-too-short          (3), -- proposed pdu size too short
        refused-by-the-VDE-Handler  (4)  -- VAA creation impossible or not allowed
        },
load-data-set               [7]     IMPLICIT  ENUMERATED {
        --data set load services error
        other                       (0),
        primitive-out-of-sequence   (1), -- according to the DataSet loading state transitions
        not-loadable                (2), -- loadable attribute set to FALSE
        dataset-size-too-large      (3), -- evaluated Data Set size too large
        not-awaited-segment         (4), -- proposed segment not awaited
        interpretation-failure      (5), -- segment interpretation error
        storage-failure             (6), -- segment storage error
        data-set-not-ready          (7) -- Data Set not in correct state for uploading
        },
-- change-scope             [8]     IMPLICIT        reserved.
task                        [9]     IMPLICIT        ENUMERATED {
        -- TI services error
        other                           (0),
        no-remote-control               (1), -- Remote Control parameter set to FALSE
        ti-stopped                      (2), -- TI in stopped state
        ti-running                      (3), -- TI in running state
        ti-unusable                     (4) -- TI in unusable state
}
-- other                    [10]    IMPLICIT  ENUMERATED
}
```

A.6 Noms des objets

ObjectName::= Integer16

Un identificateur d'objet DLMS doit être unique dans le VDE. Sa structure interne est telle que définie en 4.4 de la présente norme.

ScopeOfAccessIsVde::= BOOLEAN --vde-specific:TRUE, vaa-specific:FALSE

```
initiate                [6]     IMPLICIT    ENUMERATED {
    -- initiate service error
    other                       (0),
    dlms-version-too-low        (1), -- proposed DLMS version too low
    incompatible-conformance    (2), -- proposed services not sufficient
    pdu-size-too-short          (3), -- proposed pdu size too short
    refused-by-the-VDE-Handler  (4)  -- VAA creation impossible or not allowed
    },
load-data-set           [7]     IMPLICIT    ENUMERATED {
 -- data set load services error
    other                       (0),
    primitive-out-of-sequence   (1), -- according to the DataSet loading state transitions
    not-loadable                (2), -- loadable attribute set to FALSE
    dataset-size-too-large      (3), -- evaluated Data Set size too large
    not-awaited-segment         (4), -- proposed segment not awaited
    interpretation-failure      (5), -- segment interpretation error
    storage-failure             (6), -- segment storage error
    data-set-not-ready          (7) -- Data Set not in correct state for uploading
    },
-- change-scope         [8]     IMPLICIT    reserved.
task                    [9]     IMPLICIT    ENUMERATED {
    -- TI services error
    other                       (0),
    no-remote-control           (1), -- Remote Control parameter set to FALSE
    ti-stopped                  (2), -- TI in stopped state
    ti-running                  (3), -- TI in running state
    ti-unusable                 (4) -- TI in unusable state
}
-- other                [10]    IMPLICIT    ENUMERATED
}
```

A.6. Object name

ObjectName::= Integer16

A DLMS object identifier must be unique within the VDE. Its internal structure is as defined in 4.4 of this standard.

ScopeOfAccessIsVde ::= BOOLEAN -- vde-specific : TRUE, vaa-specific : FALSE

A.7 Protocole de gestion des contextes

```
InitiateRequest::= SEQUENCE {
        dedicated-key OCTET STRING OPTIONAL,
          -- shall not be encoded in DLMS without encryption
        response-allowed BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service          [0]   IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number               Unsigned8,
        proposed-conformance                       Conformance,
        proposed-max-pdu-size                      Unsigned16
        }

InitiateResponse::= SEQUENCE {
        negotiated-quality-of-service        [0] IMPLICIT   Integer8 OPTIONAL,
        negotiated-dlms-version-number                      Unsigned8,
        negotiated-conformance                              Conformance,
        negotiated-max-pdu-size                             Unsigned16,
        vaa-name                                            ObjectName
        }

Conformance::= [APPLICATION 30] IMPLICIT BIT STRING (SIZE(16)) {
                -- the bit is set when the corresponding service or functionality is available
                get-data-set-attribute              (0),
                get-ti-attribute                    (1),
                get-variable-attribute              (2),
                read                                (3),
                write                               (4),
                unconfirmedWrite                    (5),
                change-scope                        (6),
                start                               (7),
                stop-resume                         (8),    -- Stop and Resume services
                make-usable                         (9),
                data-set-load                       (10),   -- DataSet loading services are supported
                selection-in-get-name-list          (11),   -- detailed selection supported
                detailed-access-low-bit             (12),   -- supported detailed
                detailed-access-high-bit            (13),   --  access nested level
                multiple-variable-list              (14),   -- variable list supported in variable access service
                data-set-upload                     (15)    -- support of DataSet uploading services
                }

AbortRequest::= NULL                    -- the DLMS client asks for aborting the communication
```

A.7 Context management protocol

```
InitiateRequest::= SEQUENCE {
        dedicated-key OCTET STRING OPTIONAL,
          -- shall not be encoded in DLMS without encryption
        response-allowed BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service     [0]   IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number          Unsigned8,
        proposed-conformance                  Conformance,
        proposed-max-pdu-size                 Unsigned16
        }

InitiateResponse::= SEQUENCE {
        negotiated-quality-of-service   [0]   IMPLICIT Integer8 OPTIONAL,
        negotiated-dlms-version-number        Unsigned8,
        negotiated-conformance                Conformance,
        negotiated-max-pdu-size               Unsigned16,
        vaa-name                              ObjectName
        }

Conformance::= [APPLICATION 30] IMPLICIT BIT STRING (SIZE(16)) {
              -- the bit is set when the corresponding service or functionality is available
              get-data-set-attribute          (0),
              get-ti-attribute                (1),
              get-variable-attribute          (2),
              read                            (3),
              write                           (4),
              unconfirmedWrite                (5),
              change-scope                    (6),
              start                           (7),
              stop-resume                     (8),   -- Stop and Resume services
              make-usable                     (9),
              data-set-load                   (10), -- DataSet loading services are supported
              selection-in-get-name-list      (11), -- detailed selection supported
              detailed-access-low-bit         (12), -- supported detailed
              detailed-access-high-bit        (13), -- access nested level
              multiple-variable-list          (14), -- variable list supported in variable access service
              data-set-upload                 (15)  -- support of DataSet uploading services
              }

AbortRequest::= NULL                  -- the DLMS client asks for aborting the communication
```

A.8 Protocole de support de VDE

```
Identify ::= BOOLEAN

GetStatusRequest::= Identify

GetStatusResponse::= SEQUENCE {
        vde-type                Integer16,
        serial-number           OCTET STRING,
        status                  ENUMERATED {
            ready                       (0),
            nochange                    (1),
            inoperable                  (2)
            } DEFAULT ready,
        list-of-vaa             SEQUENCE OF ObjectName,
        identify    SEQUENCE {
                    resources                       VisibleString,
                    vendor-name                     VisibleString,
                    model                           VisibleString,
                    version-number                  Unsigned8
                    } OPTIONAL

        }

GetNameListRequest::= SEQUENCE {
        lifetime-selection      [0] IMPLICIT ENUMERATED {
            vde-only                    (0),
            data-set-only               (1)
            } OPTIONAL, -- not present means : vde-and-data-set
        object-class-selection  [1] IMPLICIT ENUMERATED {
            named-variable              (0),
            named-variable-list         (1),
            message-box                 (2),
            task-invocation             (3),
            data-set                    (4),
            vaa                         (7)
            } OPTIONAL, -- not present means : all
        scope-of-access-selection   [2] IMPLICIT ScopeOfAccessIsVde OPTIONAL,
            -- not present means : vde- and all vaa-specific
        vaa-name                [3] IMPLICIT ObjectName OPTIONAL,
        continue-after          [4] IMPLICIT ObjectName OPTIONAL
        }

GetNameListResponse::= SEQUENCE {
        more-follows            BOOLEAN DEFAULT FALSE,
        list-of-object-name     SEQUENCE OF ObjectName
        -- The SEQUENCE of Object Name must be encoded exactly in the order specified by the
        -- DLMS user: the first element in the first position and so on
        }
```

A.8 VDE support protocol

```
Identify ::= BOOLEAN

GetStatusRequest::= Identify

GetStatusResponse::= SEQUENCE {
        vde-type                Integer16,
        serial-number           OCTET STRING,
        status                  ENUMERATED {
            ready                       (0),
            nochange                    (1),
            inoperable                  (2)
            } DEFAULT ready,
        list-of-vaa             SEQUENCE OF ObjectName,
        identify    SEQUENCE {
                    resources                   VisibleString,
                    vendor-name                 VisibleString,
                    model                       VisibleString,
                    version-number              Unsigned8
                    } OPTIONAL

        }

GetNameListRequest::= SEQUENCE {
        lifetime-selection          [0]   IMPLICIT ENUMERATED {
            vde-only                        (0),
            data-set-only                   (1)
            } OPTIONAL, -- not present means : vde-and-data-set
        object-class-selection      [1]   IMPLICIT ENUMERATED {
            named-variable                  (0),
            named-variable-list             (1),
            message-box                     (2),
            task-invocation                 (3),
            data-set                        (4),
            vaa                             (7)
            } OPTIONAL, -- not present means : all
        scope-of-access-selection   [2]   IMPLICIT ScopeOfAccessIsVde OPTIONAL,
            -- not present means : vde- and all vaa-specific
        vaa-name                    [3]   IMPLICIT ObjectName OPTIONAL,
        continue-after              [4]   IMPLICIT ObjectName OPTIONAL
        }

GetNameListResponse::= SEQUENCE {
        more-follows                BOOLEAN DEFAULT FALSE,
        list-of-object-name         SEQUENCE OF ObjectName
        -- The SEQUENCE of Object Name must be encoded exactly in the order specified by the
        -- DLMS user: the first element in the first position and so on
        }
```

A.9 Protocole de gestion du jeu de données

```
InitiateLoadRequest::= SEQUENCE {
        data-set-name               ObjectName,
        number-of-segments          Unsigned16
        }

DSNameAlreadyExists ::= BOOLEAN

InitiateLoadResponse ::= DSNameAlreadyExists

LoadSegmentRequest::= SEQUENCE {
        segment-number              Unsigned16,
        segment                     OCTET STRING,
        data-set-content            VisibleString OPTIONAL
        }

LoadSegmentResponse::= Unsigned16       -- specify the number of the received segment

TerminateLoadRequest::= NULL            --the DLMS client asks for achieving the load process

TerminateLoadResponse::= ENUMERATED {       -- identify the final state of the Data Set
          ready                     (0),
          empty                     (1)
          }

InitiateUpLoadRequest::= ObjectName -- the name of the Data Set to be uploaded

InitiateUpLoadResponse ::= Unsigned16 -- the number of segments

UpLoadSegmentRequest::= Unsigned16 -- the number of the requested segment

UpLoadSegmentResponse::= SEQUENCE {
        segment-number              Unsigned16,  -- specify the number of the transmitted segment
        segment                     OCTET STRING
        }

TerminateUpLoadRequest::= NULL          --the DLMS client asks for achieving the load process

TerminateUpLoadResponse::= NULL         --the final state of the DataSet is always READY

GetDataSetAttributeRequest::= ObjectName            -- identify the Data Set

GetDataSetAttributeResponse::= SEQUENCE {
        scope-of-access         [0]   IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1]   IMPLICIT BOOLEAN DEFAULT FALSE,
        data-set-content        [2]   IMPLICIT VisibleString,
        loadable                [3]   IMPLICIT BOOLEAN DEFAULT FALSE,
        list-of-ti              [4]   IMPLICIT SEQUENCE OF ObjectName,
        uploading               [5]   IMPLICIT BOOLEAN DEFAULT FALSE,
```

A.9 Data set management protocol

```
InitiateLoadRequest::= SEQUENCE {
        data-set-name               ObjectName,
        number-of-segments          Unsigned16
        }

DSNameAlreadyExists ::= BOOLEAN

InitiateLoadResponse ::= DSNameAlreadyExists

LoadSegmentRequest::= SEQUENCE {
        segment-number              Unsigned16,
        segment                     OCTET STRING,
        data-set-content            VisibleString OPTIONAL
        }

LoadSegmentResponse::= Unsigned16       -- specify the number of the received segment

TerminateLoadRequest::= NULL            --the DLMS client asks for achieving the load process

TerminateLoadResponse::= ENUMERATED {       -- identify the final state of the Data Set
          ready                     (0),
          empty                     (1)
          }

InitiateUpLoadRequest::= ObjectName -- the name of the Data Set to be uploaded

InitiateUpLoadResponse ::= Unsigned16 -- the number of segments

UpLoadSegmentRequest::= Unsigned16 -- the number of the requested segment

UpLoadSegmentResponse::= SEQUENCE {
        segment-number              Unsigned16,  -- specify the number of the transmitted segment
        segment                     OCTET STRING
        }

TerminateUpLoadRequest::= NULL          --the DLMS client asks for achieving the load process

TerminateUpLoadResponse::= NULL         --the final state of the DataSet is always READY

GetDataSetAttributeRequest::= ObjectName -- identify the Data Set

GetDataSetAttributeResponse::= SEQUENCE {
        scope-of-access             [0]  IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change            [1]  IMPLICIT BOOLEAN DEFAULT FALSE,
        data-set-content            [2]  IMPLICIT VisibleString,
        loadable                    [3]  IMPLICIT BOOLEAN DEFAULT FALSE,
        list-of-ti                  [4]  IMPLICIT SEQUENCE OF ObjectName,
        uploading                   [5]  IMPLICIT BOOLEAN DEFAULT FALSE,
```

```
state       [6] CHOICE {
      ready                              [0]     IMPLICIT NULL,
      loading                            [1]     IMPLICIT SEQUENCE {
            current-segment                              Unsigned16,
            number-of-segments                           Unsigned16
            },
      empty                              [2]     IMPLICIT NULL
      } DEFAULT ready NULL
}
```

A.10 Protocole de gestion de VAA

```
ChangeScopeRequest::= SEQUENCE {
      new-scopeScopeOfAccessIsVde DEFAULT TRUE,
      new-vaa                 ObjectName OPTIONAL, -- name of the target VAA
       -- This case shall be selected only when transferring objects from the requesting VAA to another
      changed-scope-objects   SEQUENCE OF ObjectName OPTIONAL
       -- Absence of this parameter means that a change in executive-VAA is requested.
      }

ChangeScopeResponse ::= SEQUENCE OF CHOICE {      -- Partial success allowed
          new-scope-OK          [0]  IMPLICIT NULL,
          scope-change-failure  [1]  IMPLICIT ChangeScopeFailure,
      }

ChangeScopeFailure ::= ENUMERATED {
          unknown-error                 (0),
          scope-may-not-change          (1),
          refused-by-VDE-handler        (2),
          object-undefined              (3),
          class-inconsistent            (4)
          }
```

```
state       [6] CHOICE {
      ready                                 [0]     IMPLICIT NULL,
      loading                               [1]     IMPLICIT SEQUENCE {
            current-segment                                 Unsigned16,
            number-of-segments                              Unsigned16
            },
      empty                                 [2]     IMPLICIT NULL
      } DEFAULT ready NULL
}
```

A.10 VAA management protocol

```
ChangeScopeRequest::= SEQUENCE {
      new-scopeScopeOfAccessIsVde DEFAULT TRUE,
      new-vaa                   ObjectName OPTIONAL, -- name of the target VAA
       -- This case shall be selected only when transferring objects from the requesting VAA to another
      changed-scope-objects     SEQUENCE OF ObjectName OPTIONAL
       -- Absence of this parameter means that a change in executive-VAA is requested.
      }

ChangeScopeResponse ::= SEQUENCE OF CHOICE {        -- Partial success allowed
          new-scope-OK          [0]   IMPLICIT NULL,
          scope-change-failure  [1]   IMPLICIT ChangeScopeFailure,
      }

ChangeScopeFailure ::= ENUMERATED {
          unknown-error                 (0),
          scope-may-not-change          (1),
          refused-by-VDE-handler        (2),
          object-undefined              (3),
          class-inconsistent            (4)
          }
```

A.11 Protocole de gestion de l'invocation de tâches

```
StartRequest::= SEQUENCE {
        ti-name                 ObjectName,
        more-details            BIT STRING OPTIONAL
        }

StartResponse::= NULL                   -- the task is in the RUNNING state

StopRequest::= ObjectName               -- identify the task to stop

StopResponse::= NULL                    -- the task is in the STOPPED state

ResumeRequest::= ObjectName             -- identify the task to resume

ResumeResponse::= NULL                  -- the task is in the RUNNING state

MakeUsableRequest::= ObjectName         -- identify the task to be made usable

MakeUsableResponse::= NULL              -- the task is in the STOPPED state

GetTIAttributeRequest::= ObjectName     -- identify the task

GetTIAttributeResponse::= SEQUENCE {
        scope-of-access         [0]   IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1]   IMPLICIT BOOLEAN DEFAULT FALSE,
        remote-control-enabled  [2]   IMPLICIT BOOLEAN DEFAULT TRUE,
        state                   [3]   IMPLICIT ENUMERATED {
              unusable                      (0),
              running                       (1),
              stopped                       (2)
              } DEFAULT unusable
        }
```

A.11 Task invocation management protocol

```
StartRequest::= SEQUENCE {
        ti-name                 ObjectName,
        more-details            BIT STRING OPTIONAL
        }

StartResponse::= NULL                   -- the task is in the RUNNING state

StopRequest::= ObjectName               -- identify the task to stop

StopResponse::= NULL                    -- the task is in the STOPPED state

ResumeRequest::= ObjectName             -- identify the task to resume

ResumeResponse::= NULL                  -- the task is in the RUNNING state

MakeUsableRequest::= ObjectName         -- identify the task to be made usable

MakeUsableResponse::= NULL              -- the task is in the STOPPED state

GetTIAttributeRequest::= ObjectName     -- identify the task

GetTIAttributeResponse::= SEQUENCE {
        scope-of-access         [0]     IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1]     IMPLICIT BOOLEAN DEFAULT FALSE,
        remote-control-enabled  [2]     IMPLICIT BOOLEAN DEFAULT TRUE,
        state                   [3]     IMPLICIT ENUMERATED {
                unusable                        (0),
                running                         (1),
                stopped                         (2)
                } DEFAULT unusable
        }
```

A.12 Protocole d'accès aux variables

```
TypeDescription::= CHOICE {
                -- Complex data
        array                   [1] IMPLICIT        SEQUENCE {
                number-of-elements              Unsigned16,
                type-description                TypeDescription
                },
        structure               [2] IMPLICIT        SEQUENCE OF TypeDescription,
                -- Simple Data
        boolean                 [3] IMPLICIT        NULL,
        bit-string              [4] IMPLICIT        NULL,
        double-long             [5] IMPLICIT        NULL,
        double-long-unsigned    [6] IMPLICIT        NULL,
        floating-point          [7] IMPLICIT        NULL,
        octet-string            [9] IMPLICIT        NULL,
        visible-string          [10] IMPLICIT       NULL,
        time                    [11] IMPLICIT       NULL,
        bcd                     [13] IMPLICIT       NULL,
        integer                 [15] IMPLICIT       NULL,
        long                    [16] IMPLICIT       NULL,
        unsigned                [17] IMPLICIT       NULL,
        long-unsigned           [18] IMPLICIT       NULL }

DetailedAccess ::= OCTET STRING

-- Note (normative) : the Detailed Access is subject to a special, optimized, encoding based on the
following abstract syntax description.

Optimized-DetailedAccess::= SEQUENCE OF CHOICE {
        -- Tags 0 to 2 are for "detailed-access-selected"
        component-with-det-access   [0]     IMPLICIT SEQUENCE {
                                    item                Unsigned8,
                                    detailed-access     DetailedAccess
                                    },
        index-with-det-access       [1]     IMPLICIT SEQUENCE {
                                    item                Unsigned8,
                                    detailed-access     DetailedAccess
                                    },
        index-range-with-det-access [2]     IMPLICIT SEQUENCE {
                                    start-index         Unsigned8,
                                    number-of-index     Unsigned8,
                                    detailed-access     DetailedAccess
                                    },
        -- Tags 3 to 5 are for "access-selected"
        component                   [3]     IMPLICIT Unsigned8,
        index                       [4]     IMPLICIT Unsigned8,
        index-range                 [5]     IMPLICIT SEQUENCE {
                start-index                                 Unsigned8,
                number-of-index                             Unsigned8
                }
        }
```

A.12 Variable access protocol

```
TypeDescription::= CHOICE {
        -- Complex data
    array                       [1] IMPLICIT    SEQUENCE {
        number-of-elements              Unsigned16,
        type-description                TypeDescription
        },
    structure                   [2] IMPLICIT    SEQUENCE OF TypeDescription,
        -- Simple Data
    boolean                     [3] IMPLICIT    NULL,
    bit-string                  [4] IMPLICIT    NULL,
    double-long                 [5] IMPLICIT    NULL,
    double-long-unsigned        [6] IMPLICIT    NULL,
    floating-point              [7] IMPLICIT    NULL,
    octet-string                [9] IMPLICIT    NULL,
    visible-string              [10] IMPLICIT   NULL,
    time                        [11] IMPLICIT   NULL,
    bcd                         [13] IMPLICIT   NULL,
    integer                     [15] IMPLICIT   NULL,
    long                        [16] IMPLICIT   NULL,
    unsigned                    [17] IMPLICIT   NULL,
    long-unsigned               [18] IMPLICIT   NULL }

DetailedAccess ::= OCTET STRING

-- Note (normative) : the Detailed Access is subject to a special, optimized, encoding based on the following
abstract syntax description.

Optimized-DetailedAccess::= SEQUENCE OF CHOICE {
    -- Tags 0 to 2 are for "detailed-access-selected"
    component-with-det-access   [0]     IMPLICIT SEQUENCE {
                                item                Unsigned8,
                                detailed-access     DetailedAccess
                                },
    index-with-det-access       [1]     IMPLICIT SEQUENCE {
                                item                Unsigned8,
                                detailed-access     DetailedAccess
                                },
    index-range-with-det-access [2]     IMPLICIT SEQUENCE {
                                start-index         Unsigned8,
                                number-of-index     Unsigned8,
                                detailed-access     DetailedAccess
                                },
    -- Tags 3 to 5 are for "access-selected"
    component                   [3]     IMPLICIT Unsigned8,
    index                       [4]     IMPLICIT Unsigned8,
    index-range                 [5]     IMPLICIT SEQUENCE {
        start-index                                 Unsigned8,
        number-of-index                             Unsigned8
        }
    }
```

-- Dans la forme optimisée de l'encodage utilisée par Detailed Access, seuls les tags et les valeurs sont codés (à l'exclusion des longueurs, qui se déduisent du fait que les valeurs individuellement sont toutes de type Unsigned8). Plus formellement, chaque composant successif du Detailed Access est décrit par deux champs: Type et Contenu.

Le champ Type, codé sur un octet, contient, dans sa moitié de poids faible, la valeur du tag du composant ; dans son bit poids fort, il indique s'il s'agit du dernier composant de la SEQUENCE à laquelle il appartient (dans ce cas, ce bit est positionné à 0).

Si le tag vaut 2 ou 5, le champ Contenu a une longueur de deux octets et contient la valeur des deux indexes de la sous-séquence.; sinon, sa longueur est d'un octet et son contenu est la valeur du simple index de la sous-séquence.

Premier octet Champ Type					Deuxième octet Champ Contenu	Troisième octet Champ Contenu (suite)
poids fort				poids faible		
bit 7	..	bit 3	..	bit 0		

Premier octet:

bit 7: indicateur du dernier composant de la séquence

0: dernier composant

1: encore des composants après

bit 3 à bit 0: valeur du tag

Deuxième octet:

range du composant dans la structure, commençant à 0;

n° de l'élément dans le tableau, commençant à 1.

Troisième octet:

optionnel (utile si la valeur du tag est 2 ou 5) pour le second index de la sous-séquence.

-- In the optimized form of encoding used for the Detailed Access, only the Tags and the Values are encoded (excluding the lengths, deducible from the fact that all individual values are of Unsigned8 type). More formally, each successive component of the Detailed Access is described by two fields : Type and Content.

The Type field coded in an octet contains in its first half (least significant) the tag of the component. In its most significant bit (msb), it indicates whether it is the last component of the SEQUENCE it belongs to (in which case, this bit is set to 0).

If the Tag is 2 or 5, the Content field has a length of two octets and contains the value of the two indices of the subsequence ; otherwise, it has a length of one octet and contains the value of the single index of the subsequence.

First octet Type field	Second octet Content field	Third octet Content field (cntd)
msb .. bit 3 .. lsb bit 7 .. bit 3 .. bit 0		

First octet:
bit 7: flag of the last component of the sequence
0: last component
1: more components after
bit 3 to bit 0: tag value

Second octet:
range of the component in the structure, beginning with 0;
No. of the element in the array, beginning with 1.

Third octet:
optional (useful if tag value is 2 or 5) for the second index in the subsequence.

```
Data::= CHOICE {
                -- Complex data
    array                       [1] IMPLICIT        SEQUENCE OF Data,
    structure                   [2] IMPLICIT        SEQUENCE OF Data,
                -- Simple Data
    boolean                     [3] IMPLICIT        BOOLEAN,
    bit-string                  [4] IMPLICIT        BIT STRING,
    double-long                 [5] IMPLICIT        Integer32,
    double-long-unsigned        [6] IMPLICIT        Unsigned32,
    floating-point              [7] IMPLICIT        OCTET STRING,
    octet-string                [9] IMPLICIT        OCTET STRING,
    visible-string              [10] IMPLICIT       VisibleString,
    time                        [11] IMPLICIT       GeneralizedTime,
    bcd                         [13] IMPLICIT       Integer8,
    integer                     [15] IMPLICIT       Integer8,
    long                        [16] IMPLICIT       Integer16,
    unsigned                    [17] IMPLICIT       Unsigned8,
    long-unsigned               [18] IMPLICIT       Unsigned16,
    compact-array               [19] IMPLICIT       SEQUENCE {
        contents-description                [0]     TypeDescription,
        array-contents                      [1] IMPLICIT OCTET STRING
        }}


DataAccessError::= ENUMERATED {
    hardware-fault                  (1),
    temporary-failure               (2),
    read-write-denied               (3),
    object-undefined                (4),
    object-class-inconsistent       (9),
    object-unavailable              (11),
    type-unmatched                  (12),
    scope-of-access-violated        (13)
    }


VariableAccessSpecification::= CHOICE {
    variable-name           [2]     IMPLICIT ObjectName,
    detailed-access         [3]     IMPLICIT SEQUENCE {
        variable-name                       ObjectName,
        detailed-access                     DetailedAccess
        }
    }


ReadRequest::= SEQUENCE OF VariableAccessSpecification

ReadResponse::= SEQUENCE OF CHOICE {
    data                    [0]     Data,
    data-access-error       [1]     IMPLICIT DataAccessError
    }
```

```
Data::= CHOICE {
            -- Complex data
    array                   [1] IMPLICIT    SEQUENCE OF Data,
    structure               [2] IMPLICIT    SEQUENCE OF Data,
            -- Simple Data
    boolean                 [3] IMPLICIT    BOOLEAN,
    bit-string              [4] IMPLICIT    BIT STRING,
    double-long             [5] IMPLICIT    Integer32,
    double-long-unsigned    [6] IMPLICIT    Unsigned32,
    floating-point          [7] IMPLICIT    OCTET STRING,
    octet-string            [9] IMPLICIT    OCTET STRING,
    visible-string          [10] IMPLICIT   VisibleString,
    time                    [11] IMPLICIT   GeneralizedTime,
    bcd                     [13] IMPLICIT   Integer8,
    integer                 [15] IMPLICIT   Integer8,
    long                    [16] IMPLICIT   Integer16,
    unsigned                [17] IMPLICIT   Unsigned8,
    long-unsigned           [18] IMPLICIT   Unsigned16,
    compact-array           [19] IMPLICIT   SEQUENCE {
        contents-description                [0]     TypeDescription,
        array-contents                      [1] IMPLICIT OCTET STRING
        }}


DataAccessError::= ENUMERATED {
    hardware-fault                  (1),
    temporary-failure               (2),
    read-write-denied               (3),
    object-undefined                (4),
    object-class-inconsistent       (9),
    object-unavailable              (11),
    type-unmatched                  (12),
    scope-of-access-violated        (13)
    }


VariableAccessSpecification::= CHOICE {
    variable-name       [2]     IMPLICIT ObjectName,
    detailed-access     [3]     IMPLICIT SEQUENCE {
        variable-name                   ObjectName,
        detailed-access                 DetailedAccess
        }
    }


ReadRequest::= SEQUENCE OF VariableAccessSpecification

ReadResponse::= SEQUENCE OF CHOICE {
    data                [0]     Data,
    data-access-error   [1]     IMPLICIT DataAccessError
    }
```

```
WriteRequest::= SEQUENCE {
        variable-access-specification          SEQUENCE OF VariableAccessSpecification,
        list-of-data                           SEQUENCE OF Data
        }

WriteResponse::= SEQUENCE OF CHOICE {
        success                 [0]            IMPLICIT NULL,
        data-access-error       [1]            IMPLICIT DataAccessError
        }

UnconfirmedWriteRequest::= SEQUENCE {
        variable-access-specification          SEQUENCE OF VariableAccessSpecification,
        list-of-data                           SEQUENCE OF Data
        }

InformationReportRequest::= SEQUENCE {
        current-time                           GeneralizedTime OPTIONAL,
        variable-access-specification          SEQUENCE OF VariableAccessSpecification,
        list-of-data                           SEQUENCE OF Data
        }

GetVariableAttributeRequest::= ObjectName

GetVariableAttributeResponse::= CHOICE {
        named-variable          [0]            IMPLICIT NamedVariable,
        named-variable-list     [1]            IMPLICIT NamedVariableList,
        message-box             [2]            IMPLICIT MessageBox
        }

NamedVariable::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        type-description        [3]          TypeDescription,
        read-write-flag         [4] IMPLICIT BOOLEAN DEFAULT TRUE,
        available               [5] IMPLICIT BOOLEAN DEFAULT TRUE
        }

NamedVariableList::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        list-of-named-variables [6] IMPLICIT SEQUENCE OF ObjectName
        }

MessageBox::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        read-write-flag         [4] IMPLICIT BOOLEAN DEFAULT TRUE,
        available               [5] IMPLICIT BOOLEAN DEFAULT TRUE
        }
```

```
WriteRequest::= SEQUENCE {
        variable-access-specification       SEQUENCE OF VariableAccessSpecification,
        list-of-data                        SEQUENCE OF Data
        }

WriteResponse::= SEQUENCE OF CHOICE {
        success                 [0]         IMPLICIT NULL,
        data-access-error       [1]         IMPLICIT DataAccessError
        }

UnconfirmedWriteRequest::= SEQUENCE {
        variable-access-specification       SEQUENCE OF VariableAccessSpecification,
        list-of-data                        SEQUENCE OF Data
        }

InformationReportRequest::= SEQUENCE {
        current-time                        GeneralizedTime OPTIONAL,
        variable-access-specification       SEQUENCE OF VariableAccessSpecification,
        list-of-data                        SEQUENCE OF Data
        }

GetVariableAttributeRequest::= ObjectName

GetVariableAttributeResponse::= CHOICE {
        named-variable          [0]         IMPLICIT NamedVariable,
        named-variable-list     [1]         IMPLICIT NamedVariableList,
        message-box             [2]         IMPLICIT MessageBox
        }

NamedVariable::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        type-description        [3]          TypeDescription,
        read-write-flag         [4] IMPLICIT BOOLEAN DEFAULT TRUE,
        available               [5] IMPLICIT BOOLEAN DEFAULT TRUE
        }

NamedVariableList::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        list-of-named-variables [6] IMPLICIT SEQUENCE OF ObjectName
        }

MessageBox::= SEQUENCE {
        scope-of-access         [0] IMPLICIT ScopeOfAccessIsVde DEFAULT TRUE,
        scope-may-change        [1] IMPLICIT BOOLEAN DEFAULT FALSE,
        life-time-is-vde        [2] IMPLICIT BOOLEAN DEFAULT TRUE,
        read-write-flag         [4] IMPLICIT BOOLEAN DEFAULT TRUE,
        available               [5] IMPLICIT BOOLEAN DEFAULT TRUE
        }
```

Annexe B
(informative)

Description des états utilisateurs DLMS

B.1 Généralités

La présente annexe décrit les états que peut prendre un utilisateur DLMS ainsi que les changements d'états.

B.1.1 Introduction

Les services décrits dans les articles précédents sont divisés en deux groupes :

- dans le premier, des services qui se limitent à collecter des information auprès du VDE. Il n'y a pas de changement d'état ;

- dans le second, si le service a fonctionné avec succès, le VDE passe d'un état initial à un état final différent de l'état initial.

Dans cette annexe, seuls les services du second groupe seront décrits par leurs changements d'état.

Tous les changements d'états autorisés sont décrits dans cette annexe. Un service qui tente un changement d'état qui n'est pas décrit n'est pas valable et une réponse d'erreur de type "STATE-VIOLATION" (non respect de l'état) doit être émise. Cette restriction est faite afin de garantir la cohérence du modèle DLMS.

B.1.2 Conventions de description

Tous les tableaux de changement d'état doivent être considérées du point de vue du prestataire DLMS.

Les primitives de services auxquelles on a attaché un "+" contiennent un paramètre Result(+). Les primitives de services auxquelles on a attaché un "-" contiennent un paramètre Result(-).

Annex B
(informative)

DLMS user states description

B.1 Overview

This annex describes the states in which a DLMS user may enter and the transitions between these states.

B.1.1 Introduction

The services described in the clauses above are divided in two groups:

- in the first one, the services just pick up some information at the VDE. No state transition occurs;

- in the second one, and if the service succeeds, the VDE transits from an initial state into a different final state.

In this annex, only the services of the second group are described by their state transitions.

All the allowed state transitions are described in this annex. A service that attempts an un-described transition is invalid and a negative response with the error type "STATE-VIOLATION" must be issued. This restriction is made in order to ensure state coherency of DLMS model.

B.1.2 Descriptive convention

All the state transition tables have to be seen from the viewpoint of the DLMS provider.

Service primitives to which a "+" is appended contain a Result(+) parameter. Service primitives to which a "-" is appended contain a Result(-) parameter.

B.2 Gestion des états du contexte

B.2.1 Tableau de changement d'état

Le présent paragraphe définit les changements d'état quand on entre ou quand on sort de l'environnement DLMS. L'état initial pour le client et le serveur utilisateurs de DLMS est NoCon (pas de contexte). Les deux tableaux sont considérés du point de vue de l'utilisateur DLMS, un pour le client et l'autre pour le serveur DLMS.

Tableau B.1 - Changements d'état du contexte du serveur

Etat Initial	Evénements	Actions	Etat final
NoCon	Initiate.ind and Initiate() = Ok	Create_VAA() Initiate.rsp(+, Version, Conformance, VAA_name, PDU_size)	IDLE
NoCon	Initiate.ind and Initiate() <> Ok	Initiate.rsp (-)	NoCon
IDLE	Initiate.ind and Initiate() = Ok	Initiate.rsp(+, Version, Conformance, VAA_name, PDU_size)	IDLE
IDLE	Initiate.ind and Initiate() <> Ok	Initiate.rsp(-)	IDLE
IDLE	Abort.ind	Delete_VAA()	NoCon

B.2 Context management states

B.2.1 State transition tables

This subclause defines the state transitions for entering and leaving the DLMS environment. The initial state for both the client and the server DLMS users is NoCon (No Context). The two tables are seen from the point of view of a DLMS user, one for the DLMS client and one for the DLMS server.

Table B.1 - Server context state transitions

Initial state	Events	Actions	Final state
NoCon	Initiate.ind and Initiate() = Ok	Create_VAA() Initiate.rsp(+, Version, Conformance, VAA_name, PDU_size)	IDLE
NoCon	Initiate.ind and Initiate() <> Ok	Initiate.rsp (-)	NoCon
IDLE	Initiate.ind and Initiate() = Ok	Initiate.rsp(+, Version, Conformance, VAA_name, PDU_size)	IDLE
IDLE	Initiate.ind and Initiate() <> Ok	Initiate.rsp(-)	IDLE
IDLE	Abort.ind	Delete_VAA()	NoCon

Tableau B.2 - Changements d'état du contexte du client

Etat initial	Evénements	Actions	Etat final
NoCon	Initiate_Event	Initiate.req(List of Version, Conformance, PDU_size)	CON.W
CON.W	Initiate.cnf(-)	none	NoCon
CON.W	Initiate.cnf(+)	none	IDLE
CON.W	Abort_Event	Abort.req()	NoCon
IDLE	Abort_Event	Abort.req()	NoCon
IDLE	Initiate_Event	Initiate.req(List of Version, Conformance, PDU_size)	CON.W
CON.W	Initiate.cnf	none	IDLE

B.2.2 Description des événements

Les primitives de service agissant comme des événements sont décrites dans les articles précédents. Pour des besoins de clarification, seuls les paramètres nécessaires à une pleine compréhension sont cités explicitement.

L'événement Initiate_Event (initialiser un événement) caractérise une décision du client utilisateur de DLMS d'initialiser une AA (association d'application) avec le serveur utilisateur de DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

Événement Abort_Event (abandonner un événement) caractérise une décision du client d'abandonner une AA avec le serveur utilisateur de DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

B.2.3 Description des actions

La fonction Initiate() (initialiser) est une fonction interne du VDE-handler (gestionnaire VDE). Elle renvoie Ok si le VDE-handler accepte la proposition d'Initiate. Les règles pour accepter ou refuser sont décrites dans l’article 5.

La fonction Create_VAA() (Créer une VAA) est une fonction interne du VDE-handler. Elle crée un objet VAA associé à un client DLMS s'il n'existe pas déjà.

Table B.2 - Client context state transitions

Initial state	Events	Actions	Final state
NoCon	Initiate_Event	Initiate.req(List of Version, Conformance, PDU_size)	CON.W
CON.W	Initiate.cnf(-)	none	NoCon
CON.W	Initiate.cnf(+)	none	IDLE
CON.W	Abort_Event	Abort.req()	NoCon
IDLE	Abort_Event	Abort.req()	NoCon
IDLE	Initiate_Event	Initiate.req(List of Version, Conformance, PDU_size)	CON.W
CON.W	Initiate.cnf	none	IDLE

B.2.2 Event description

The service primitives which act as events are described in the clauses above. For clarification purposes, only the parameters which are necessary to understanding the whole are explicitly given.

The Initiate_Event event characterizes the decision of the DLMS client to initiate an AA with the server DLMS user. The basis of this event is outside the scope of this specification.

The Abort_Event event characterizes the decision of the DLMS client to abort an AA with the DLMS server. The basis of this event is outside the scope of this specification.

B.2.3 Action description

The Initiate() function is an internal function of the VDE-handler. It returns Ok if the VDE-handler accepts the Initiate proposition. The rules for accepting or refusing are described in clause 5.

The Create_VAA() function is an internal function of the VDE-handler. It creates the VAA object associated with the DLMS client if it does not already exist.

La fonction Delete_VAA() (effacer une VAA) est une fonction interne du VDE-handler. Elle efface l'objet VAA associé au client DLMS.

B.2.4. Description des états

L'état NoCon (pas de contexte) est l'état initial d'un utilisateur DLMS. Tant qu'il est dans cet état, un client ne peut qu'émettre la primitive Initiate.req (demande d'initialisation).

Tant qu'il est dans l'état CON.W (attente de contexte), le client DLMS ne peut qu'émettre une primitive Abort.req (demande d'abandon).

L'état requis pour utiliser les services de DLMS est l'état IDLE (inactif). Cet état se subdivise en un ensemble de sous-états qui sont décrits dans les paragraphes ci-dessous. Tous les états définis dans ces paragraphes sont des sous-états de l'état IDLE.

Les seuls événements qui peuvent provoquer une sortie de l'état IDLE sont Abort_Event ou la réception d'une primitive Abort.ind (indication d'abandon).

B.3 Gestion des états du jeu de données

B.3.1 Tableaux de changement d'état

B.3.1.1 Chargement

Le présent paragraphe définit les changements d'état pour le chargement du jeu de données. Les deux tableaux sont considérés du point de vue de l'utilisateur DLMS, un pour le client l'autre pour le serveur DLMS.

L'état du serveur DLMS est IDLE et les sous-états sont définis en fonction de l'attribut Data Set State (état du jeu de données). Le sous-état initial d'un serveur DLMS peut être soit Empty (vide) soit Ready (prêt).

The Delete_VAA() function is an internal function of the VDE-handler. It deletes the VAA object associated with the DLMS client.

B.2.4 State description

The NoCon (No Context) state is the initial state of a DLMS user. While in this state, a DLMS client may only issue the initiate.req primitive.

While in the state CON.W (Context Waited), the DLMS client may only issue an Abort.req primitive.

The required state for using DLMS services is the IDLE state. This state is divided into a set of substates which are described in the following subclauses. All the states defined in the following subclauses are substates of the IDLE state.

The only events which cause an exit from the IDLE state are the Abort_Event or the receiption of an Abort.ind primitive.

B.3 Data set management states

B.3.1 State transition tables

B.3.1.1 Loading

This subclause defines the state transitions for loading the data set. The two tables are seen from the point of view of a DLMS user, one for the DLMS client and one for the DLMS server.

The state for the DLMS server is IDLE and substates are defined according to the Data Set State attribute. The initial substate of the DLMS server may be either Empty or Ready.

Tableau B.3 - Changements d'état du chargement du jeu de données du serveur

Etat initial	Evénements	Actions	Etat final
Empty	InitiateLoad.ind(DS_name, NbSeg) and InitiateLoad() = Ok	DataSet.name = DS_name DataSet.state = LOADING DataSet.last_seg = NbSeg DataSet.cur_seg = 0 InitiateLoad.rsp(+)	Loading
Empty	InitiateLoad.ind() and InitiateLoad() <> Ok	InitiateLoad.rsp(-)	Empty
Ready	InitiateLoad.ind(DS_name, NbSeg) and InitiateLoad() = Ok	Delete_DS_Lifetime_objects() Unuse_DS_TI() DataSet.name = DS_name DataSet.state = LOADING DataSet.last_seg = NbSeg DataSet.cur_seg = 0 InitiateLoad.rsp(+)	Loading
Ready	InitiateLoad.ind() and InitiateLoad() <> Ok	InitiateLoad.rsp(-)	Ready
Loading	LoadSegment.ind(SegNb , Segment) and SegNb = DataSet.cur_seg + 1 or SegNb = DataSet.cur_seg) and Process(Segment) = Ok	DataSet.cur_seg = SegNb LoadSegment.rsp(DataSet.cur_seg)	Loading
Loading	LoadSegment.ind(SegNb , Segment) and (SegNb <> DataSet.cur_seg + 1 and SegNb <> DataSet.cur_seg) or Process(Segment) <>Ok)	LoadSegment.rsp(-)	Loading
Loading	TerminateLoad.ind() and DataSet.cur_seg = DataSet.last_seg	DataSet.state = READY TerminateLoad.rsp(+)	Ready
Loading	TerminateLoad.ind() and DataSet.cur_seg <> DataSet.last_seg	DataSet.state = EMPTY Delete_DS_Lifetime_objects() Unuse_DS_TI() TerminateLoad.rsp(+)	Empty

Table B.3 - Server data set loading state transitions

Initial state	Events	Actions	Final state
Empty	InitiateLoad.ind(DS_name, NbSeg) and InitiateLoad() = Ok	DataSet.name = DS_name DataSet.state = LOADING DataSet.last_seg = NbSeg DataSet.cur_seg = 0 InitiateLoad.rsp(+)	Loading
Empty	InitiateLoad.ind() and InitiateLoad() <> Ok	InitiateLoad.rsp(-)	Empty
Ready	InitiateLoad.ind(DS_name, NbSeg) and InitiateLoad() = Ok	Delete_DS_Lifetime_objects() Unuse_DS_TI() DataSet.name = DS_name DataSet.state = LOADING DataSet.last_seg = NbSeg DataSet.cur_seg = 0 InitiateLoad.rsp(+)	Loading
Ready	InitiateLoad.ind() and InitiateLoad() <> Ok	InitiateLoad.rsp(-)	Ready
Loading	LoadSegment.ind(SegNb , Segment) and SegNb = DataSet.cur_seg + 1 or SegNb = DataSet.cur_seg) and Process(Segment) = Ok	DataSet.cur_seg = SegNb LoadSegment.rsp(DataSet.cur_seg)	Loading
Loading	LoadSegment.ind(SegNb , Segment) and (SegNb <> DataSet.cur_seg + 1 and SegNb <> DataSet.cur_seg) or Process(Segment) <>Ok)	LoadSegment.rsp(-)	Loading
Loading	TerminateLoad.ind() and DataSet.cur_seg = DataSet.last_seg	DataSet.state = READY TerminateLoad.rsp(+)	Ready
Loading	TerminateLoad.ind() and DataSet.cur_seg <> DataSet.last_seg	DataSet.state = EMPTY Delete_DS_Lifetime_objects() Unuse_DS_TI() TerminateLoad.rsp(+)	Empty

Tableau B.4 - Changements d'état du chargement du jeu de données du client

Etat initial	Evénements	Actions	Etat final
IDLE	InitiateLoad_Event	InitiateLoad.req(DS_name, NbSeg)	IL.W
IL.W	InitiateLoad.cnf(-)	none	IDLE
IL.W	InitiateLoad.cnf(+)	SegNb = 1	DLoad
DLoad	LoadSegment_Event	LoadSegment.req(SegNb, Segment)	DL.W
DLoad	TerminateLoad_Event	TerminateLoad.req()	TL.W
DL.W	LoadSegment.cnf(+, Cur_Seg)	SegNb = Cur_Seg + 1	DLoad
DL.W	LoadSegment.cnf(-)	none	DLoad
TL.W	TerminateLoad.cnf(+)	none	IDLE
TL.W	TerminateLoad.cnf(-)	none	DLoad

B.3.1.2 Rapatriement

Le présent paragraphe définit les changements d'état pour le rapatriement du jeu de données (du serveur vers le client). Les deux tableaux sont considérés du point de vue de l'utilisateur DLMS, un pour le client, l'autre pour le serveur DLMS.

L'état du serveur DLMS est IDLE et les sous-états sont définis en fonction de l'attribut Data Set State (état du jeu de données). Les sous-états initial d'un serveur DLMS doit être Ready (prêt).

Table B.4 - Client data set loading state transitions

Initial state	Events	Actions	Final state
IDLE	InitiateLoad_Event	InitiateLoad.req(DS_name, NbSeg)	IL.W
IL.W	InitiateLoad.cnf(-)	none	IDLE
IL.W	InitiateLoad.cnf(+)	SegNb = 1	DLoad
DLoad	LoadSegment_Event	LoadSegment.req(SegNb, Segment)	DL.W
DLoad	TerminateLoad_Event	TerminateLoad.req()	TL.W
DL.W	LoadSegment.cnf(+, Cur_Seg)	SegNb = Cur_Seg + 1	DLoad
DL.W	LoadSegment.cnf(-)	none	DLoad
TL.W	TerminateLoad.cnf(+)	none	IDLE
TL.W	TerminateLoad.cnf(-)	none	DLoad

B.3.1.2 UpLoading

This subclause defines the state transitions for uploading the data set (from server to client). The two tables are seen from the point of view of a DLMS user, one for the DLMS client and one for the DLMS server.

The state for the DLMS server is IDLE and substates are defined according to the Data Set State attribute. The initial substate of the DLMS server must be Ready.

Tableau B.5 - Changements d'état du rapatriement du jeu de données du serveur

Etat initial	Evénements	Actions	Etat final
Ready	InitiateUpLoad.ind(DS_name) and InitiateUpLoad() = Ok	Uploading = TRUE InitiateUpLoad.rsp(+, NbSeg)	Ready
Ready	InitiateUpLoad.ind() and InitiateUpLoad() <> Ok	InitiateUpLoad.rsp(-)	Ready
Ready	UpLoadSegment.ind(SegNb) and UpLoadSegment() = Ok	UpLoadSegment.rsp(Segment, SegNb)	Ready
Ready	UpLoadSegment.ind(SegNb) and UpLoadSegment() <> Ok	UpLoadSegment.rsp(-)	Ready
Ready	TeminateUpLoad.ind()	Uploading = FALSE TerminateUpLoad.rsp(+)	Ready

Tableau B.6 - Changements d'état du rapatriement du jeu de données du client

Etat initial	Evénements	Actions	Etat final
IDLE	InitiateUpLoad_Event	InitiateUpLoad.req(DS_name)	IUL.W
IUL.W	InitiateUpLoad.cnf(-)	none	IDLE
IUL.W	InitiateUpLoad.cnf(+)	SegNb = 1	ULoad
ULoad	UpLoadSegment_Event	UpLoadSegment.req(SegNb)	UL.W
ULoad	TerminateUpLoad_Event	TerminateUpLoad.req()	TUL.W
UL.W	UpLoadSegment.cnf(Segment, Cur_Seg)	SegNb = Cur_Seg + 1	ULoad
UL.W	UpLoadSegment.cnf(-)	none	ULoad
TUL.W	TerminateUpLoad.cnf(+)	none	IDLE
TUL.W	TerminateUpLoad.cnf(-)	none	ULoad

B.3.2 Description des événements

Les primitives de service agissant comme des événements sont décrites dans les articles précédents. Pour des besoins de clarification, seuls les paramètres nécessaires à une pleine compréhension sont cités explicitement.

L'événement InitiateLoad_Event (initialiser le chargement) caractérise une décision du client DLMS d'initialiser le chargement du jeu de données dans le serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

Table B.5 - Server data set uploading state transitions

Initial state	Events	Actions	Final state
Ready	InitiateUpLoad.ind(DS_name) and InitiateUpLoad() = Ok	Uploading = TRUE InitiateUpLoad.rsp(+, NbSeg)	Ready
Ready	InitiateUpLoad.ind() and InitiateUpLoad() <> Ok	InitiateUpLoad.rsp(-)	Ready
Ready	UpLoadSegment.ind(SegNb) and UpLoadSegment() = Ok	UpLoadSegment.rsp(Segment, SegNb)	Ready
Ready	UpLoadSegment.ind(SegNb) and UpLoadSegment() <> Ok	UpLoadSegment.rsp(-)	Ready
Ready	TeminateUpLoad.ind()	Uploading = FALSE TerminateUpLoad.rsp(+)	Ready

Table B.6 - Client data set uploading state transitions

Initial state	Events	Actions	Final state
IDLE	InitiateUpLoad_Event	InitiateUpLoad.req(DS_name)	IUL.W
IUL.W	InitiateUpLoad.cnf(-)	none	IDLE
IUL.W	InitiateUpLoad.cnf(+)	SegNb = 1	ULoad
ULoad	UpLoadSegment_Event	UpLoadSegment.req(SegNb)	UL.W
ULoad	TerminateUpLoad_Event	TerminateUpLoad.req()	TUL.W
UL.W	UpLoadSegment.cnf(Segment, Cur_Seg)	SegNb = Cur_Seg + 1	ULoad
UL.W	UpLoadSegment.cnf(-)	none	ULoad
TUL.W	TerminateUpLoad.cnf(+)	none	IDLE
TUL.W	TerminateUpLoad.cnf(-)	none	ULoad

B.3.2 Event description

The service primitives which act as events are described in the clauses above. For clarification purposes, only the parameters which are necessary for the understanding of the whole are explicitly given.

The InitiateLoad_Event event characterizes the decision of the DLMS client to initiate the down loading of the data set in the DLMS server. The basis of this event is outside the scope of this specification.

L'événement InitiateUpLoad_Event (initialiser le rapatriement) caractérise une décision du client DLMS d'initialiser le rapatriement du jeu de données venant du serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

L'événement LoadSegment_Event (charger un segment) caractérise une décision du client DLMS de continuer le téléchargement du jeu de données dans un serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

L'événement TerminateLoad_Event (terminer le chargement) caractérise une décision du client DLMS de terminer le chargement d'un jeu de données dans un serveur DLMS. Les bases de cet événement sont en dehors du domaine de cette spécification.

L'événement UpLoadSegment_Event (rapatrier un segment) caractérise la décision du client DLMS de continuer le rapatriement d'un jeu de données venant du serveur DLMS. Les bases de cet événement sont en dehors du domaine de cette spécification.

L'événement TerminateUpLoad_Event (terminer le rapatriement) caractérise une décision du client DLMS de terminer le rapatriement d'un jeu de données du serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

B.3.3 Description des actions

La fonction InitiateLoad() (initialisation du chargement) est une fonction interne du VDE-handler (gestionnaire VDE). Elle renvoie Ok si le VDE-handler accepte la proposition de chargement. Les règles pour accepter ou refuser sont décrites dans l'article 7.

La fonction InitiateUpLoad() (initialisation du rapatriement) est une fonction interne du VDE-handler (gestionnaire VDE). Elle renvoie Ok si le VDE-handler accepte la proposition de rapatriement. Les règles pour accepter ou refuser sont décrites dans l'article 7.

La fonction Process(Segment) (traiter un segment) est une fonction interne de VDE-handler. Elle renvoie Ok si le VDE-handler a réussi le transfert et la mise en place du segment spécifié (ou l'accès au segment spécifié dans le cas d'un rapatriement).

La fonction Delete_DS_Lifetime_Objects() (détruire les objets dont la durée de vie est jeu de données) est une fonction interne du VDE-handler. Elle détruit les objets définis avec une durée de vie jeu de données.

La fonction Unuse_DS_TI() (rendre TI non utilisable) est une fonction interne du VDE-handler. Elle sert à inhiber les TI (invocation de tâches) qui utilisent le jeu de données. Toutes les TI définies dans le VDE utilisant actuellement le jeu de données passent à l'état UNUSABLE (inutilisable).

The InitiateUpLoad_Event event characterizes the decision of the DLMS client to initiate the uploading of the data set from the DLMS server. The basis of this event is outside the scope of this specification.

The LoadSegment_Event event characterizes the decision of the DLMS client to continue the down loading of the data set in the DLMS server. The basis of this event is outside the scope of this specification.

The TerminateLoad_Event event characterizes the decision of the client DLMS user to terminate the down loading of the data set in the DLMS server. The basis of this event is outside the scope of this specification.

The UpLoadSegment_Event event characterizes the decision of the DLMS client to continue the uploading of the data set from the DLMS server. The basis of this event is outside the scope of this specification.

The TerminateUpLoad_Event event characterizes the decision of the client DLMS user to terminate the uploading of the data set from the server DLMS. The basis of this event is outside the scope of this specification.

B.3.3 Action description

The InitiateLoad() function is an internal function of the VDE-handler. It returns Ok if the VDE-handler accepts the down loading proposition. The rules for accepting or refusing are described in clause 7.

The InitiateUpLoad() function is an internal function of the VDE-handler. It returns Ok if the VDE-handler accepts the uploading proposition. The rules for accepting or refusing are described in clause 7.

The Process(Segment) function is an internal function of the VDE-handler. It returns Ok if the VDE-handler succeeds in translating and in installing (or accessing in the case of an Upload) the specified segment.

The Delete_DS_Lifetime_Objects() function is an internal function of the VDE-handler. It deletes the VDE objects defined with a data set lifetime.

The Unuse_DS_TI() function is an internal function of the VDE-handler. Its purpose is to inhibit the TIs that use the data set. All the TIs defined at the VDE and currently using the data set transit into the UNUSABLE state.

La fonction UpLoadSegment() (rapatrier le segment) est une fonction interne du VDE-handler. Elle renvoie Ok si le VDE-handler arrive à accéder au segment spécifié. Les règles pour accepter ou pour refuser son décrites dans l'article 7.

B.3.4 Description des états

Les états Empty (vide), Ready (prêt) et Loading (en cours de chargement) sont des sous-états de l'état IDLE (inactif) pour les besoins du chargement du jeu de données.

L'état IL.W (initialisation du chargement en attente) indique que le client utilisateur de DLMS attend la confirmation d'une primitive InitiateLoad.req (demande d'initialisation de chargement) émise au préalable.

L'état DL.W (chargement en attente) indique que le client utilisateur de DLMS attend la confirmation d'une primitive LoadSegment.req (demande de chargement de segment) émise au préalable.

L'état DLoad (chargement en cours) est l'état normal pour qu'un client utilisateur de DLMS puisse contrôler la totalité du processus de chargement du jeu de données dans un VDE éloigné.

L'état TL.W (attente de fin de chargement) est utilisé par le client utilisateur de DLMS pour terminer la séquence de chargement.

L'état IUL.W (initialisation rapatriement en attente) indique que le client utilisateur de DLMS attend la confirmation d'une primitive InitiateUpLoad.req (demande d'initialisation de rapatriement) émise au préalable.

L'état UL.W (rapatriement d'un segment en attente) indique que le client utilisateur de DLMS attend la confirmation d'une primitive UpLoadSegment.req (demande de rapatriement de segment) émise au préalable.

L'état ULoad (rapatriement en cours) est l'état normal pour qu'un client utilisateur de DLMS puisse contrôler la totalité du processus de rapatriement du jeu de données dans un VDE éloigné.

L'état TUL.W (attente de fin de rapatriement) est utilisé par le client utilisateur de DLMS pour terminer la séquence de rapatriement.

The UpLoadSegment() function is an internal function of the VDE-handler. It returns Ok if the VDE-handler succeeds in accessing the specified segment. The rules for accepting or refusing are described in clause 7.

B.3.4 State description

The Empty, Ready and Loading states are substates of the IDLE state for the data set loading purpose of the server DLMS user.

The IL.W (Initiate Load Waiting) state indicates that the client DLMS user is waiting for a confirm of a previously issued InitiateLoad.req primitive.

The DL.W (Down Load Waiting) state indicates that the client DLMS user is waiting for a confirm of a previously issued LoadSegment.req primitive.

The DLoad (Down Loading) state is the normal state for the client DLMS user to control the whole process of the data set down loading to the distant VDE.

The TL.W (Terminate Load Waiting) state is used by the client DLMS user to terminate the down loading sequence.

The IUL.W (InitiateUpLoad Waiting) state indicates that the client DLMS user is waiting for a confirm of a previously issued InitiateUpLoad.req primitive.

The UL.W (UpLoadSegment Waiting) state indicates that the client DLMS user is waiting for a confirm of a previously issued UpLoadSegment.req primitive.

The ULoad (UpLoading) state is the normal state for the client DLMS user to control the whole process of the data set uploading to the distant VDE.

The TUL.W (Terminate UpLoad Waiting) state is used by the client DLMS user to terminate the uploading sequence.

B.4 Gestion des états de l'invocation de tâches

B.4.1 Les tableaux de changement d'état

Le présent paragraphe définit la gestion des changements d'état des TI (invocations de tâches). Les deux tableaux sont considérés du point de vue de l'utilisateur DLMS, un pour le client utilisateur de DLMS et l'autre pour le serveur utilisateur de DLMS.

L'état pour le serveur utilisateur de DLMS est IDLE (inactif) et les sous-états sont définis conformément à l'attribut TI State (état de la TI). Le sous-état initial du serveur utilisateur de DLMS peut être Running (en fonctionnement), Stopped (arrêté) ou Unusable (inutilisable).

Tableau B.7 - Gestion des changements d'état des TI du serveur

Etat initial	Evénements	Actions	Etat final
Stopped	Start.ind(param) and TI_Start(param) = Ok	TI.state = RUNNING Update_DS_ListOfTI() Start.rsp(+)	Running
Stopped	Start.ind(param) and TI_Start(param) = Recoverable	Start.rsp(-, STOPPED)	Stopped
Stopped	Start.ind(param) and TI_Start(param) = UnRecoverable	Start.rsp(-, UNUSABLE)	Unusable
Stopped	Resume.ind(param) and TI_Resume(param) = Ok	TI.state = RUNNING Update_DS_ListOfTI() Resume.rsp(+)	Running
Stopped	Resume.ind(param) and TI_Resume(param) = Recoverable	Resume.rsp(-, STOPPED)	Stopped
Stopped	Resume.ind(param) and TI_Resume(param) = UnRecoverable	Resume.rsp(-, UNUSABLE)	Unusable
Running	Stop.ind(param) and TI_Stop(param) = Ok	TI.state = STOPPED Update_DS_ListOfTI() Stop.rsp(+)	Stopped
Running	Stop.ind(param) and TI_Stop(param) = Recoverable	Stop.rsp(-, RUNNING)	Running
Running	Stop.ind(param) and TI_Stop(param) = UnRecoverable	Stop.rsp(-, UNUSABLE)	Unusable
Unusable	MakeUsable.ind(param) and TI_MakeUsable(param) = Ok	TI.state = STOPPED MakeUsable.rsp(+, STOPPED)	Stopped
Unusable	MakeUsable.ind(param) and TI_MakeUsable(param) <> Ok	MakeUsable.rsp(+, UNUSABLE)	Unusable

B.4 Task invocation management states

B.4.1 State transition tables

This subclause defines the state transitions for the TI management. The two tables are seen from the point of view of a DLMS user, one for the client DLMS user and one for the server DLMS user.

The state for the server DLMS user is IDLE and substates are defined according to the TI State attribute. The initial substates of the server DLMS user may be Running, Stopped or Unusable.

Table B.7 - Server TI management state transitions

Initial state	Events	Actions	Final state
Stopped	Start.ind(param) and TI_Start(param) = Ok	TI.state = RUNNING Update_DS_ListOfTI() Start.rsp(+)	Running
Stopped	Start.ind(param) and TI_Start(param) = Recoverable	Start.rsp(-, STOPPED)	Stopped
Stopped	Start.ind(param) and TI_Start(param) = UnRecoverable	Start.rsp(-, UNUSABLE)	Unusable
Stopped	Resume.ind(param) and TI_Resume(param) = Ok	TI.state = RUNNING Update_DS_ListOfTI() Resume.rsp(+)	Running
Stopped	Resume.ind(param) and TI_Resume(param) = Recoverable	Resume.rsp(-, STOPPED)	Stopped
Stopped	Resume.ind(param) and TI_Resume(param) = UnRecoverable	Resume.rsp(-, UNUSABLE)	Unusable
Running	Stop.ind(param) and TI_Stop(param) = Ok	TI.state = STOPPED Update_DS_ListOfTI() Stop.rsp(+)	Stopped
Running	Stop.ind(param) and TI_Stop(param) = Recoverable	Stop.rsp(-, RUNNING)	Running
Running	Stop.ind(param) and TI_Stop(param) = UnRecoverable	Stop.rsp(-, UNUSABLE)	Unusable
Unusable	MakeUsable.ind(param) and TI_MakeUsable(param) = Ok	TI.state = STOPPED MakeUsable.rsp(+, STOPPED)	Stopped
Unusable	MakeUsable.ind(param) and TI_MakeUsable(param) <> Ok	MakeUsable.rsp(+, UNUSABLE)	Unusable

Tableau B.8 - Gestion des changements d'états des TI du client

Etat initial	Evénements	Actions	Etat final
IDLE	Start_Event	Start.req(name, param)	TI.W
TI.W	Start.cnf()	none	IDLE
IDLE	Resume_Event	Resume.req(name)	TI.W
TI.W	Resume.cnf()	none	IDLE
IDLE	Stop_Event	Stop.req(name)	TI.W
TI.W	Stop.cnf()	none	IDLE
IDLE	MakeUsable_Event	MakeUsable.req(name)	TI.W
TI.W	MakeUsable.cnf()	none	IDLE

B.4.2 Description des événements

Les primitives de services agissant comme des événements sont décrites dans les articles précédents. Pour des besoins de clarification, seuls les paramètres nécessaires à une pleine compréhension sont cités explicitement.

L'événement Start_Event (démarrage d'un événement) caractérise une décision du client utilisateur de DLMS de démarrer une TI qui est actuellement arrêtée dans le serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

L'événement Resume_Event (reprise d'un événement) caractérise une décision du client utilisateur de DLMS de relancer une TI actuellement arrêtée dans le serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

L'événement Stop_Event (arrêt d'un événement) caractérise une décision d'un client utilisateur de DLMS d'arrêter une TI active dans le serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

L'événement MakeUsable_Event (rendre utilisable) caractérise la décision du client utilisateur de DLMS de rendre utilisable une TI actuellement dans l'état UNUSABLE (inutilisable) dans le serveur DLMS. Les bases de cet événement sont en dehors du domaine de la présente spécification.

Table B.8 - Client TI management state transitions

Initial state	Events	Actions	Final state
IDLE	Start_Event	Start.req(name, param)	TI.W
TI.W	Start.cnf()	none	IDLE
IDLE	Resume_Event	Resume.req(name)	TI.W
TI.W	Resume.cnf()	none	IDLE
IDLE	Stop_Event	Stop.req(name)	TI.W
TI.W	Stop.cnf()	none	IDLE
IDLE	MakeUsable_Event	MakeUsable.req(name)	TI.W
TI.W	MakeUsable.cnf()	none	IDLE

B.4.2 Event description

The service primitives which act as events are described in the clauses above. For clarification purposes, only the parameters which are necessary for the understanding of the whole are explicitly given.

The Start_Event event characterizes the decision of the client DLMS user to start a TI that is currently stopped at the server DLMS user. The basis of this event is outside the scope of this specification.

The Resume_Event event characterizes the decision of the client DLMS user to resume a TI that is currently stopped at the server DLMS user. The basis of this event is outside the scope of this specification.

The Stop_Event event characterizes the decision of the client DLMS user to stop a TI that currently runs at the server DLMS user. The basis of this event is outside the scope of this specification.

The MakeUsable_Event event characterizes the decision of the client DLMS user to make usable a TI that is currently in the UNUSABLE state at the server DLMS user. The basis of this event is outside the scope of this specification.

B.4.3 Description des actions

La fonction TI_Start() (démarrer une TI) est une fonction interne du VDE-handler (gestionnaire VDE). Elle tente de faire passer la TI à l'état RUNNING (en fonctionnement). Elle renvoie Ok si le VDE-handler réalise la transition, Recoverable (récupérable) si le VDE-handler échoue ; la TI retourne à l'état STOPPED (arrêté), et UnRecoverable (irrécupérable) est retourné si le VDE-handler échoue définitivement. Les règles pour accepter ou refuser sont décrites en 9.2.

La fonction TI_Resume() (relancer TI) est une fonction interne du VDE-handler. Elle tente de faire passer la TI à l'état RUNNING (en fonctionnement). Elle renvoie Ok si le VDE-handler réalise le changement, Recoverable (récupérable) si le VDE-handler a échoué mais que la TI retourne à l'état STOPPED (arrêté), UnRecoverable (irrécupérable) si le VDE-handler échoue définitivement. Les règles pour accepter ou refuser sont décrites en 9.4.

La fonction TI_Stop() (arrêter TI) est une fonction interne du VDE-handler. Elle tente de faire passer une TI à l'état STOPPED (arrêté). Elle renvoie Ok si le VDE-handler réalise le changement, Recoverable si le VDE-handler échoue mais que la TI retourne à l'état RUNNING (en fonctionnement) , et UnRecoverable (irrécupérable) si le VDE-handler échoue définitivement. Les règles pour accepter ou refuser son décrites en 9.3.

La fonction TI_MakeUsable() (rendre utilisable) est une fonction interne du VDE-handler. Elle tente de faire passer la TI de l'état UNUSABLE (inutilisable) à l'état STOPPED (arrêté). Elle renvoie Ok si le VDE-handler réalise le changement. Les règles pour accepter ou refuser sont décrites en 9.5.

La fonction Update_DS_ListOfTI() (mettre à jour liste de TI) est une fonction interne du VDE-handler. Elle sert à mettre à jour la liste des TI actuellement utilisées par le jeu de données.

B.4.4 Description des états

Les états Running (actif), Stopped (arrêté) et Unusable (inutilisable) sont des sous-états de l'état IDLE (inactif) pour les besoins de la gestion des TI du serveur utilisateur de DLMS.

L'état TI.W (invocation de tâches en attente) indique que le client utilisateur de DLMS attend la confirmation d'une des primitives Start.req (demande de démarrage), Resume.req (demande de reprise) Stop.req (demande d'arrêt) ou MakeUsable.req (demande de rendre utilisable) émises au préalable.

B.4.3 Action description

The TI_Start() function is an internal function of the VDE-handler. It attempts to make the TI transit into the RUNNING state. It returns Ok if the VDE-handler achieves the transition, Recoverable if the VDE-handler failed; the TI returns to the STOPPED state and UnRecoverable is returned if the VDE-handler definitely failed. The rules for accepting or refusing are described in 9.2.

The TI_Resume() function is an internal function of the VDE-handler. It attempts to make the TI transit into the RUNNING state. It returns Ok if the VDE-Handler achieves the transition, Recoverable if the VDE-handler failed but the TI return to the STOPPED state and UnRecoverable if the VDE-handler definitely failed. The rules for accepting or refusing are described in 9.4.

The TI_Stop() function is an internal function of the VDE-handler. It attempts to make the TI transit into the STOPPED state. It returns Ok if the VDE-handler achieves the transition, Recoverable if the VDE-handler failed but the TI returns to the RUNNING state and UnRecoverable if the VDE-handler definitely failed. The rules for accepting or refusing are described in 9.3.

The TI_MakeUsable() function is an internal function of the VDE-handler. It attempts to make the TI transit from the UNUSABLE state into the STOPPED state. It returns Ok if the VDE-handler achieves the transition. The rules for accepting or refusing are described in 9.5.

The Update_DS_ListOfTI() function is an internal function of the VDE-handler. Its purpose is to update the list of the TI currently used by the data set.

B.4.4 State description

The Running, Stopped and Unusable states are substates of the IDLE state for the TI Management purpose of the server DLMS user.

The TI.W (Task Invocation Waiting) state indicates that the client DLMS user is waiting for a confirm of a previously issued Start.req, Resume.req, Stop.req or MakeUsable.req primitive.

ICS 29.240.20 ; 33.200

Typeset and printed by the IEC Central Office
GENEVA, SWITZERLAND