INTERNATIONAL STANDARD

ISO/IEC 9040

Second edition
1997-06-15

Information technology — Open Systems Interconnection — Virtual Terminal Basic Class Service

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — Service de classe de base de terminal virtuel

Reference number
ISO/IEC 9040:1997(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 9040 was prepared by Joint Technical Committee ISO/IEC JTC1, *Information technology*, Subcommittee SC21, *Open systems interconnection, data management and open distributed processing.*

This second edition cancels and replaces the first edition (ISO 9040:1990), which has been technically revised. It also incorporates Amendment 2:1992, Technical Corrigendum 1:1991, Technical Corrigendum 2:1992 and Technical Corrigendum 3:1993.

Annexes A and C form an integral part of this International Standard. Annexes B and D are for information only.

Introduction

This International Standard is one of a set of standards produced to facilitate the interconnection of computer systems. It is related to other International Standards in the set as defined in the Reference Model for Open Systems Interconnection (ISO/IEC 7498-1). The Reference Model subdivides the area of standardization into a series of layers of specification, each of manageable size.

The purpose of this International Standard (ISO/IEC 9040) is to define the service provided in the Application Layer by the Virtual Terminal (VT) Basic Class Service.

The Virtual Terminal Basic Class Service is provided by the Virtual Terminal Basic Class Protocol specified in ISO/IEC 9041 and making use of services available from the Association Control Service Element (ACSE) in the Application Layer and the Presentation Service.

This page intentionally left blank

Information technology – Open Systems Interconnection – Virtual Terminal Basic Class Service

1 Scope

This International Standard defines, in an abstract way, the externally visible Basic Class Virtual Terminal Service within the OSI Application Layer in terms of

a) a model defining the interaction between users of the service;

b) the primitive actions and events of the service;

c) the parameter data associated with each primitive action and event;

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

The service defined in this International Standard is that which is provided by the OSI Basic Class Virtual Terminal Protocol (in conjunction with the Association Control Service Element and the Presentation Service) and which may be used by any user including other Application Service Elements. The relationship between the standards for Virtual Terminal Service, Virtual Terminal Protocol, ACSE, Presentation Layer Service and the user of the Virtual Terminal Service is shown in figure 1.

This International Standard also defines two standard default virtual terminal environment profiles and describes the form of registered virtual terminal environment profiles and control objects. Virtual terminal environment profiles define sets of virtual terminal environment parameters for use in the establishment of virtual terminal associations and subsequent negotiation. This International Standard also defines a structure

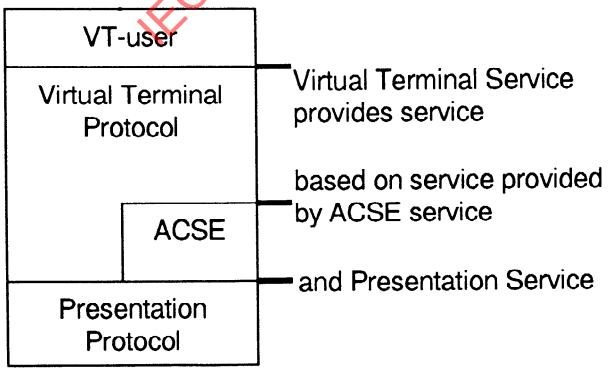


Figure 1 - Relationship of this International Standard to other OSI Application Layer Standards

of ASN.1 Object Identifiers for the objects defined in this International Standard and for use in a register of virtual terminal objects.

This International Standard does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces within a computer system. There is, therefore, no requirement for conformance to this International Standard.

This International Standard applies to interactive applications requiring terminal oriented communication expressed in terms of the transmission and manipulation of graphical images having the following characteristics:

e) the images are composed of character-box graphic elements organised into a one, two or three dimensional structure;

f) attributes may be associated with any graphic element to qualify its mode of display.

Control information for the communication can be modelled using virtual terminal control objects, and multiple devices can be modelled using virtual terminal device objects linked to the other virtual terminal objects.

2 Normative references

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

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*

ISO/IEC 2022:1994, *Information technology – Character code structure and extension techniques.*

ISO/IEC 2375:1985, *Data processing – Procedure for registration of escape sequences.*

ISO/IEC 6429:1992, *Information technology – Control functions for coded character sets.*

ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model : The Basic Model.*

ISO/IEC 8649:1996, *Information technology – Open Systems Interconnection – Service definition for Association Control Service Element.*

ISO/IEC 8824:1990, *Information technology – Open Systems Interconnection – Specification for 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 9041-1:1997, *Information technology – Open Systems Interconnection – Virtual Terminal Basic Class Protocol – Part 1: Specification.*

ISO/IEC 9834-4:1991, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities – Part 4: Register of VTE Profiles*

ISO/IEC 9834-5:1991, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities – Part 5: Register of VT Control Object Definitions*

ISO/IEC 10731:1994, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the Definition of OSI Services.*

The International Register of Coded Character Sets to be used with Escape Sequences. [1)]

3 Definitions

3.1 Global OSI definitions

This International Standard is based on the concepts developed in ISO/IEC 7498-1 and makes use of the following terms defined in it:

a) application entity;
b) Application Layer;
c) service data unit;
d) service access point.

It also makes use of the following terms defined in ISO/IEC 10731:

e) service primitive;
f) service provider;
g) primitive;
h) request (primitive);
i) indication (primitive);
j) response (primitive);
k) confirm (primitive);
l) confirmed service;
m) non-confirmed service;
n) provider-initiated service.

3.2 Association Control Service Element definitions

This International Standard makes use of the following terms defined in ISO/IEC 8649:

a) application association;
b) application entity title;
c) application control service element (ACSE).

3.3 Virtual Terminal Service definitions

For the purposes of this International Standard, the following definitions apply:

3.3.1 VT-user: A user of the Virtual Terminal Service.

3.3.2 Application VT-user: The unique VT-user which can update the FDCO; if either VT-user can update this control object then neither VT-user has this designation.

3.3.3 Terminal VT-user: If one VT-user has the designation Application VT-user then the peer VT-user has the designation Terminal VT-user.

3.3.4 character-repertoire: A set of objects which can be represented by primary attribute values; one such object, represented by its primary attribute value, can occupy an array element in a display object when the character-repertoire is in use for that array element. A control object of character-string category also has an associated repertoire.

3.3.5 character-box graphic element: An atomic element of a character-repertoire where use of the repertoire has been agreed through negotiation by the VT-users.

3.3.6 primary attribute: The attribute of an array element of a display object which is a coded representation of the character-box graphic element assigned to that array element.

3.3.7 secondary attribute: The secondary attributes of an array element comprise the character-repertoire, see 3.3.4, and the rendition attributes.

3.3.8 rendition attributes: Those secondary attributes of an array element which qualify the character-box graphic element and provide information specifying how it is intended to be presented.

3.3.9 explicit modal default: The value for a secondary attribute, defined in a VTE, which is used by the text operation to update an array element if no other value is provided or already present; may also be used by the erase operation.

1) Available from the European Computer Manufacturers Association (ECMA), 114 rue du Rhône, CH-1204 Genève, Switzerland.

3.3.10 display object: An abstract object, defined in this International Standard, for modelling the exchange of graphic information. It consists of a number of components, see 13.1.

3.3.11 array element: That part of a display object which can hold one character-box graphic element including its primary and secondary attribute values.

3.3.12 primitive display pointer: A set of one to three coordinate values which identify a particular element in a display object.

3.3.13 extended display pointer: A set of two to four coordinate values which identify a particular array element in a block defined on a display object.

3.3.14 display pointer: Used to refer to either the primitive display pointer or the extended display pointer; whether or not blocks are in use determines which is implied.

3.3.15 logical pointer: A set of two or three coordinate values which identify a particular array element in a field defined on a display object.

NOTE – The primitive display pointer and extended display pointer do not both exist simultaneously. However, when a logical pointer exists, it is in addition to either a display pointer or an extended display pointer.

3.3.16 control object: An abstract object, of a type defined in generic terms in this International Standard, for modelling the exchange of unstructured information.

NOTE – The primary application of a control object is for modelling the exchange of information of a control nature, as understood by the VT-users; the VT Service does not constrain the interpretation of this information.

3.3.17 device object: An abstract object used to model certain logical characteristics of real devices, and to link the various objects of a virtual terminal environment together and/or to real devices.

3.3.18 object updating device: A real device capable of generating values which (possibly after undergoing a transformation) are used by one of the peer VT-users to update either a display object or control object (or possibly both).

3.3.19 VT-association: An application association between two peer VT-users.

3.3.20 VT-environment (VTE): A set of parameters that together define the data structuring and operational characteristics for a particular VT-association. The VTE exists only during the lifetime of that VT-association. The parameters of the set are mutually related by a directed graph structure. The VTE may be modified during the existence of the VT-association by negotiation.

3.3.21 current-VTE: The single VTE which exists during the Data Handling phase or the Negotiation Quiescent phase; in the Data Handling phase it is a full-VTE whereas in the Negotiation Quiescent phase it is not a full-VTE.

3.3.22 draft-VTE: The VTE, if any, under negotiation. During negotiation, the draft-VTE is not necessarily a full-VTE.

3.3.23 VTE-parameter: An individual parameter of a VTE. Each VTE-parameter is given a unique name in the service which is used as the identifier for the VTE-parameter.

3.3.24 full-VTE: A VTE that is a complete directed graph of VTE-parameters in which all node parameters and terminal leaf parameters implied by all existing nodes from the root of the tree have values.

3.3.25 VT-context-value: A collective term for the set of object instances, their assigned values and the current-VTE for a particular VT-association. A VT-context-value exists only during the lifetime of the VT-association and is normally changing continuously during this time interval.

3.3.26 reset-context: The VT-context-value which will result after a VT-BREAK service. This context value is the context after the last successful current-VTE establishment; all objects will have their initial values. If no full-VTE has been established, there is no reset-context.

3.3.27 WAVAR access-right: An access-right which can be held by at most one VT-user at any time. It is used to ensure that control and display objects cannot be updated by both VT-users simultaneously.

3.3.28 access-rule: A characteristic defined for an object in a VTE which determines which VT-users can update the object at a particular time.

3.3.29 net-effecting: The conversion of a sequence of items, representing the content of one or more update operations (see 24.3), into a different, usually shorter sequence, which results in the same final states of the objects being updated.

3.3.30 concatenation: The connection of a sequence of queued update items (see 24.3) to form a single, new, queued update item.

3.3.31 segmentation: The division of a single, queued update item (see 24.3) into a sequence of new, queued update items.

3.3.32 A-mode (Asynchronous mode): A mode of operation using two display objects, one of which is updatable by the VT-user which initiated the VT-association and the other by the peer VT-user.

3.3.33 S-mode (Synchronous mode): A mode of operation using one two-way-alternate dialogue supporting one display object; at any time, the display object may only be updated by the single VT-user which owns the WAVAR access-right.

3.3.34 service: A distinct part of the total VT Service that is composed of a sequence of primitives taken from the set {request primitive, indication primitive, response primitive, confirm primitive}.

3.3.35 sequenced service: A Service for which an indication (or confirm) primitive resulting from a corresponding request (or response) primitive is initiated in sequence with all previously initiated sequenced indications (or confirms) and their corresponding requests (or responses).

3.3.36 non-sequenced service: A Service for which an indication (or confirm) primitive resulting from a corresponding request (or response) primitive is not necessarily initiated in sequence with all previously initiated indications (or confirmations) and their corresponding requests (or responses).

3.3.37 conditionally sequenced service: A Service for which

a) certain values for parameters of the service primitives result in sequenced operation, and

b) other values for parameters of the service primitives result in non-sequenced operation.

3.3.38 destructive service: A service that may cause the loss of information conveyed in previously initiated services without notification of this loss to either VT-user. Only non-sequenced services may be destructive, but not all non-sequenced services are destructive.

3.3.39 non-destructive service: A service that does not cause the loss of information conveyed in previously initiated services without notification to the VT-users.

3.3.40 service parameter: A parameter defined as part of a primitive within a VT service.

3.3.41 update-window: A mechanism associated with display object addressing which defines a range of coordinate values for an array below which an update operation cannot be performed. The absolute coordinate values in the range may increase in value during operation, but cannot decrease.

3.3.42 update-window-size: A positive, non-zero integer that defines the number of contiguous array elements within an update-window.

3.3.43 trigger: Where a control object has the trigger characteristic, any update to that control object causes delivery of queued updates and, in S-mode, transfers the WAVAR access–right to the peer VT-user.

3.3.44 VTE-profile: A pre-defined set of VTE-parameter values making up a VTE.

NOTE – Some VTE-profiles are parameterised such that values for profile arguments must be supplied by the VT-users.

3.3.45 default VTE-profile: A VTE-profile, defined in annex A, that is used to establish a VTE in the absence of a VT-user specified VTE-profile at VT-association establishment.

3.3.46 registered VTE-profile: A VTE-profile registered in a register of VT Objects administered by a Registration Authority established as defined in ISO/IEC 9834-4. It has assigned to it a unique ASN.1 OBJECT IDENTIFIER value.

3.3.47 registered control object: A control object registered in a register of VT Objects administered by a Registration Authority established as defined in ISO/IEC 9834-5. It has assigned to it a unique ASN.1 OBJECT IDENTIFIER value.

3.3.48 privately defined VTE-profile: A VTE-profile whose use has been agreed privately by means outside the scope of this International Standard; it is neither a default VTE-profile nor a registered VTE-profile.

3.3.49 VTE-profile argument: An argument of a parameterised VTE-profile which must be given a value to enable a full-VTE to be made from the VTE-profile. It can be a normal VTE-profile argument or a special VTE-profile argument. Each registered VTE-profile defines the applicable arguments and their semantics.

3.3.50 normal VTE-profile argument: A VTE-profile argument which corresponds exactly to a VTE-parameter and assumes the same identifier. A VTE-profile may also specify that other VTE-parameters are also defined by the value of such an argument.

3.3.51 special VTE-profile argument: A VTE-profile argument which does not correspond to a VTE-parameter (i.e., is not from the directed graph and has no standard identifier) but has a valid use as defined in the VTE-profile definition.

3.3.52 multiple interaction negotiation (MIN): A process which enables a draft-VTE to be modified or extended in stages to create a new full-VTE acceptable to the service provider and both VT-users.

3.3.53 MIN-initiator: The VT-user that initiated the VT-START-NEG service which was successful in causing the transition to Negotiation Active phase.

3.3.54 MIN-acceptor: The VT-user that accepted the VT-START-NEG service which was successful in causing the transition to Negotiation Active phase.

3.3.55 block: A structure element of the display object. A block is a rectangular area which supports relative addressing (relative to the origin of the block) for a sub-area within a Y-array.

3.3.56 field: A structure element of the display object. A field provides one-dimensional logical addressing over a designated subset of the array elements of a Y-array of a display object.

3.3.57 field-element: A rectangular area of a Y-array of a display object forming part of a field.

3.3.58 Field Definition Control Object (FDCO): A type of control object which holds the definition of fields for a display object as Field Definition Records.

3.3.59 Field Definition Record (FDR): Holds the status and definition of one field.

3.3.60 Field Entry Instruction Control Object (FEICO): A type of control object which holds data entry instructions as Field Entry Instruction Records.

3.3.61 Field Entry Pilot Control Object (FEPCO): A type of control object which holds data entry pilots as Field Entry Pilot Records.

3.3.62 Field Entry Instruction Record (FEIR): Holds a set of data entry rules as Field Entry Instructions (FEIs).

3.3.63 Field Entry Pilot Record (FEPR): Holds a data entry pilot consisting of field entry event, field entry conditions and a sequence of field entry reactions.

3.3.64 Field Entry Instruction (FEI): Applies a rule for controlled data entry into a field.

3.3.65 Field Entry Event (FEE): An event which may occur during controlled data entry and cause an entry reaction.

3.3.66 Field Entry Condition (FEC): A condition of the data entry which taken with an entry event predicates an entry reaction.

3.3.67 Field Entry Reaction (FER): A reaction to an entry event and entry condition defined in terms of operations on other objects in the virtual terminal service.

3.3.68 Transmission Policy Control Object (TPCO): A type of control object which holds variable values which determine how field contents are transmitted after a controlled data entry.

3.3.69 Context Control Object (CCO): A type of control object which provides the local context of the VT-user at certain stages of the data entry dialogue.

3.3.70 Reference Information Object (RIO): A container for information, separate from any display object, which can be referred to during a communication.

3.3.71 Termination Conditions Control Object (TCCO): A type of control object which holds termination conditions effective on the operation of one or more device objects linked to it.

3.3.72 termination-event: A locally defined atomic event related to input from an object updating device (for example, entering a character, pressing a function key, setting a flag) which the VT-users agree (through negotiation) to designate as causing input data to be delivered to the peer VT-user.

3.3.73 ripple: Ripple is a mechanism whereby the contents of array elements of the Display Object may be moved into adjacent array elements without the need of reconveying the contents from one VT-user to the other.

3.3.74 Ripple Mode Control Object (RMCO): a type of control object associated with the ripple mechanism which controls the operation of the ripple, for example, the extent of the ripple.

3.3.75 ripple-extent: A ripple-extent is that part of a DO within which the ripple mechanism operates.

3.3.76 copy buffer: A copy buffer stores the content and structure of an extent of the Display Object to allow subsequent transfer of this structure and content to some other extent of the DO.

3.3.77 extended-y-array: The set of array elements which are within the currently defined blocks of a particular y-array.

3.3.78 extended-z-array: The set of array elements which are within the currently defined blocks of some y-array of the z-array.

3.3.79 filling: Is the operation which defines the values of the primary and secondary attributes of character box elements which are left undefined as the result of ripple operations.

3.3.80 ripple coordinate: Determines the coordinate direction of the ripple, i.e., the coordinate which is altered as the result of ripple operations, and hence determines the unit of ripple. It takes a value from the set ("x", "y", "z", "k").

3.3.81 ripple direction: Specifies whether units are to be moved in a forward (increasing coordinate) or a backward (decreasing coordinate) direction, in a ripple operation.

3.3.82 unit of ripple: Determines the units which the ripple operations work upon. It takes a value from the set ("array element", "x-array", "y-array").

4 Abbreviations

4.1 General

ACSE	Association Control Service Element
ASN.1	Abstract Syntax Notation One
BNF	Backus-Naur Form
ECMA	European Computer Manufacturers Association
IRV	International Reference Version (of ISO/IEC 646)
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MIN	Multiple Interaction Negotiation
OSI	Open Systems Interconnection
QOS	Quality of Service
VT	Virtual Terminal
VTE	Virtual Terminal Environment

4.2 Modes of operation

A-Mode	Asynchronous Mode
S-Mode	Synchronous Mode

4.3 VTE model components

ACS	Access Control Store
CCA	Conceptual Communication Area
CCO	Context Control Object
CDS	Conceptual Data Store
CO	Control Object
CSS	Control, Signal and Status store
DSD	Data Structure Definition
DO	Display Object
ECO	Echo Control Object
FDCO	Field Definition Control Object
FDR	Field Definition Record
FEC	Field Entry Condition
FEE	Field Entry Event
FEI	Field Entry Instruction
FEICO	Field Entry Instruction Control Object
FEIR	Field Entry Instruction Record

FEPCO	Field Entry Pilot Control Object
FEPR	Field Entry Pilot Record
FER	Field Entry Reaction
RIO	Reference Information Object
RMCO	Ripple Mode Control Object
TCCO	Termination Conditions Control Object
TCO	Termination Control Object
TPCO	Transmission Policy Control Object

4.4 Access-rules

NSAC	Not Subject to Access Control
WACA	Write Access Connection (VT-association) Acceptor
WACI	Write Access Connection (VT-association) Initiator
WAVAR	Write Access VARiable

5 Conventions

This International Standard uses the descriptive conventions contained in the ISO/IEC Service Conventions (ISO/IEC 10731).

Additional conventions used are

a) a character string enclosed in angle brackets, i.e. <xxx>, denotes the equivalent ISO/IEC 646 mnemonic;

b) a character string of the form n...N, where n is usually 0 or 1, denotes the set of integers greater than or equal to n, i.e., no limit is defined in this International Standard;

c) an informal notation analogous to ASN.1 is used in places, with apparently similar terms; where exact ASN.1 notation or terminology is intended, the usage will be introduced by reference to ASN.1;

d) a syntax derived from Backus-Naur Form (BNF) is used for VTE-profile definitions; this is explained in annex A.

6 General features

Clauses 6 and 7 give an Overview of the Virtual Terminal Service defined in this International Standard.

6.1 Introduction

The Basic Class Virtual Terminal Service supports the interactive transfer and manipulation of graphic data by VT-users. This graphic data is structured in a manner which models the class of character-box oriented terminals. The basic structuring of graphic elements is limited to images consisting of character-box graphic elements arranged in a one, two or three dimensional array. Optional functional units provide additional structuring capabilities. Two modes of operation are defined for Basic Class, S-mode and A-mode.

6.2 Features of the Virtual Terminal Basic Class Service

The Virtual Terminal Basic Class Service offers the following services to the VT-user:

a) the means to establish a VT-association between two peer VT-users for the purpose of enabling virtual terminal information exchange;

b) the means to negotiate the VT functional units required;

c) the means to negotiate a consistent set of VTE-parameters;

d) the means to transfer and manipulate structured data in a way that is independent of the local representation of information used by each VT-user and that is independent of the way in which supporting communications media are used;

e) the means to control the integrity of the communication;

f) the means to terminate the VT-association either unilaterally or by mutual agreement;

g) the means to support either synchronous (S-mode) or asynchronous (A-mode) operation between the VT-users;

h) the means to exchange priority information to gain the immediate attention of a VT-user;

i) the means to terminate information transfer destructively and resynchronize the activity of the VT provider;

j) a facility for defining blocks in a display object [Blocks functional unit];

k) a facility for defining fields in a display object [Fields functional unit, also uses feature in n)];

l) additional optional access-rules for control objects in S-mode [Enhanced Access-rules functional unit];

m) means to control the asymmetry inherent in typical use of these features [uses the feature in l)];

n) a facility for defining control objects with content consisting of multiple data elements or a single partially-updateable structured data element [Structured Control Objects functional unit];

o) a facility for controlling data entry to fields using new standard types of control object [uses the feaure in n)];

p) a facility for storing and using update information in Reference Information Objects [RIOs functional unit];

q) a facility for establishing a VT-association with the capability to switch between the modes of operation when the VTE is changed;

r) display object update with ripple;

s) the reporting of exception conditions by the VT-service-provider;

t) the selective retention of VT-context between negotiation of successive VTEs.

6.3 VT Environment (VTE) and VTE-parameters

The transfer and manipulation of graphic data takes place within a VT-environment (VTE) defined by a logically consistent set of VTE-parameter values. Certain VTE-parameters are related in that a value for one VTE-parameter can constrain the existence of or permissible values for another VTE-parameter. These relationships are partly expressed by a directed graph for the VTE-parameters and partly by explicit definition.

One and only one full-VTE exists during data transfer. A full-VTE may be modified or replaced if negotiation facilities are available.

6.4 Virtual Terminal Environment Profiles

The Basic Class VT Service provides for the use of virtual terminal environment profiles (VTE-profiles) which are sets of VTE-parameters and VTE-parameter values for use in the negotiation of the VTE (see clause 11 and annex A).

6.5 Dialogue Control

The integrity of VT communication is maintained by Dialogue Control service facilities (by applying access-rules, delivery control and delivery acknowledgement, see clauses 9 and 24), and service primitive sequencing rules which provide integrity by detecting and resolving collisions (see clause 27).

7 Communication facilities

7.1 Establishment facility

The Establishment facility provides a service that establishes a VT-association and an initial VTE for that VT-association when a VT-user first invokes the Basic Class VT Service.

7.2 Termination facility

The Termination facility provides services which enable a VT-user to terminate a VT-association either in an orderly and non-destructive manner or in an immediate and potentially destructive manner. It also provides a service which enables the VT service provider to notify the VT-users when a VT-association is terminated in a potentially destructive way.

7.3 Negotiation facility

The Negotiation facility provides services which enable peer VT-users to select, modify and replace the current-VTE.

An initial VTE is established during VT-association establishment based on the VTE-profile specified. If the VT-user does not specify a VTE-profile during VT-association establishment, a default VTE-profile is used to establish the initial VTE. This VTE may subsequently be modified or replaced depending on the type of negotiation facilities available.

The type of negotiation facility available to the VT-users is determined by the VT functional units selected during VT-association establishment. In addition to the VTE-profile selection facility available during VT-association establishment, there are two other types of negotiation, i.e., switch profile negotiation and multiple interaction negotiation.

7.4 Data Transfer facility

The Data Transfer facility provides a service which enables a VT-user to update the contents of display and/or control objects to which the VT-user is currently permitted update access.

7.5 Delivery Control facility

The Delivery Control facility provides services which enable a VT-user to control, synchronise and optionally request acknowledgement of, the release to the peer VT-user of updates entered previously using the Data Transfer facility. This includes the (negotiable) ability to "quarantine" (i.e., hold back) such updates from such release until the release is requested.

7.6 Dialogue Management facility

In S-mode, the Dialogue Management facility enables the VT-users to request or cede ownership of the WAVAR access-right.

In A-mode, the Dialogue Management facility is not available.

7.7 Interrupt facilities

The Interrupt facilities are available in both modes of operation but can only be used in the Data Handling phase. There is a destructive interrupt facility and a non-destructive interrupt facility.

The **destructive interrupt** facility allows a VT-user to interrupt a previously initiated sequence of updates to display and control objects, discard all updates currently being exchanged and resume exchanging updates after the VT-service-providers have resynchronized their activities.

The **non-destructive interrupt** facility allows VT-users to exchange priority information but without destroying non-priority information; it can be used as an "attention" signalling mechanism (via appropriately defined control objects, see clause 14).

7.8 Exception Reporting facility

The Exception Reporting facility provides services which allow the VT-service-provider to report abnormal conditions to the VT-users without causing termination of the VT-association.

8 Modes of operation

Clauses 8, 9, 10 and 11 define general aspects of the Virtual Terminal Service.

The Virtual Terminal Basic Class Service supports two modes of operation. Each mode is characterised by its form of dialogue control. The initial mode of operation is selected when the VT-association is established. The ability to change modes on the establishment of a new VTE is also selected when the VT-association is established.

8.1 S-mode

S-mode (synchronous mode) has the following characteristics:

a) one or other VT-user may own the WAVAR access-right; ownership may be passed between the two VT-users;

b) only a single display object is supported with a single two-way-alternate dialogue. Update access to this display object is controlled by the WAVAR access-rule, see clause 9;

c) the VT-users may define and make use of termination condition parameters in device objects;

d) the access-rules available for any control object are as in clause 9, table 1.

8.2 A-mode

A-mode (asynchronous mode) has the following characteristics:

a) the WAVAR access-right is not available;

b) two display objects are supported, each with a monologue. One display object has access-rule WACI and can only be updated by the VT-user which initiated the VT-association; the other display object has access-rule WACA and can only be updated by the VT-user which accepted the VT-association;

c) the VT-users may define and make use of echo control objects and termination condition parameters in device objects;

d) the access-rules available for any control object are as in clause 9, table 1;

e) Dialogue Management service primitives do not apply.

9 Access-rules

Each display object is assigned an access-rule which determines whether and when each VT-user is permitted to update that display object. Each control object is assigned an access-rule or combination of access-rules which determine whether and when each VT-user is permitted to update that control object. The possible access-rules are:

- **no-access** : neither VT-user may update the object;
- **WACI** : the object may be updated only by the VT-user which initiated the VT-association;
- **WACA** : the object may be updated only by the VT-user which accepted the VT-association;
- **WAVAR** : the object may be updated only by the VT-user owning the WAVAR access-right;
- **WAVAR & WACI** : the object may be updated only by the VT-user which initiated the VT-asociation and only when it owns the WAVAR access-right;
- **WAVAR & WACA** : the object may be updated only by the VT-user which accepted the VT-asociation and only when it owns the WAVAR access-right;
- **NSAC** : the object may be updated by either VT-user at any time.

In S-mode, the single display object always has access-rule WAVAR. In A-mode, one display object has access-rule WACI and the other has access-rule WACA.

NOTES

1 A display object cannot have an access-rule which is a combination of multiple access rights. However, other special access conditions can apply to a display object, e.g., due to data entry conditions applied by special control objects.

2 Use of the extended access-rules given by the Enhanced Access-rules functional unit is independent of actual use of the other functional units although certain of the other functional units may require this functional unit to be present.

The access-rules which may be assigned to control objects in each mode are listed in table 1.

In S-mode, most services may only be initiated when the WAVAR access-right is held, see 27.4. Communication facilities are available which allow a VT-user to request or cede ownership of the WAVAR access-right, see clause 33.

Table 1 – VT Access-rule and Mode correspondence with Enhanced Access-rules functional unit

Access-rules	S-mode	A-mode
no-access	(1) YES	(1) YES
WACI	(1) YES	YES (2)
WACA	(1) YES	YES (2)
WAVAR	YES (2)	NO
WAVAR&WACI	(1) YES	NO
WAVAR&WACA	(1) YES	NO
NSAC	YES	YES

NOTES
1 Available with Enhanced Access-rules functional unit.
2 Display object can only have a value from this set.

10 VT functional units

The Basic Class Virtual Terminal Service provides a number of optional capablilities called Functional Units. The VT functional units required are selected during VT association establishment.

The optional functional units available in the VT service are:

a) Switch Profile Negotiation;
b) Multiple Interaction Negotiation;
c) Negotiated Release;
d) Urgent Data;
e) Break;
f) Enhanced Access-rules;
g) Structured Control Objects;
h) Blocks;
i) Fields;
j) Reference Information Objects;
k) Ripple;
l) Exceptions;
m) Context Retention.

The Multiple Interaction Negotiation functional unit can be selected only if the Switch Profile Negotiation functional unit is also selected.

Although formally independent, use of the Fields functional unit requires a control object which requires the Structured Control Objects functional unit.

The Exceptions functional unit can be selected only if the Break functional unit is selected.

All provisions in this International Standard which are not part of one of the optional functional units are part of the Kernel functional unit and are always available.

10.1 Negotiation functional units

The Virtual Terminal Basic Class Service provides negotiation services to enable the creation and modification of the single VTE in a manner acceptable to the service provider and both VT-users. Negotiation is available as part of the establishment facility. Additional negotiation services are optionally available (dependent on the functional units selected during association establishment) which provide two forms of negotiation, a single interaction switch profile negotiation and a multiple interaction negotiation.

At VT-association establishment, a current-VTE is established using the VTE-profile-based negotiation function embedded in the VT-ASSOCIATE service. This current-VTE may or may not be a full-VTE. If the initiating VT-user does not specify a VTE-profile with the VT-ASSOCIATE (to supply the basis for negotiating the current-VTE), the service selects a default VTE-profile appropriate to the mode of operation, see annex A. Depending on the functional units selected, this current-VTE may or may not be subsequently modified using negotiation services. If the initial current-VTE is not a full-VTE, it is necessary to use the negotiation facility to establish a full-VTE before the Data Handling phase may be entered.

The negotiation services do not modify the current-VTE unless the negotiation reaches a satisfactory conclusion. During an instance of negotiation, agreements on VTE-parameter values are recorded in a temporary VTE definition known as the draft-VTE. In the case of single interaction profile switch negotiation, a draft-VTE only has a transitory existence. However, for multiple interaction negotiation, the concept is more significant, i.e., the draft-VTE contains the new VTE as it is built up over an extended time interval.

10.1.1 Switch Profile Negotiation

The single interaction Switch Profile Negotiation is provided by a single confirmed service, VT-SWITCH-PROFILE. This form of negotiation is similar to that provided in VT-ASSOCIATE and takes the form of a proposal made by one VT-user using a VTE-profile and, where required, offered VTE-profile argument values. The VTE-profile chosen is not negotiable but offered VTE-profile argument values may be adjusted by both the service provider and the accepting VT-user (in that order) to obtain an agreed set of VTE-parameter values, but only within the range of freedom offered by the service initiator. Adjustment of VTE-parameter values by this service is constrained to those parameters defined using VTE-profile arguments in the VTE-profile specification.

10.1.2 Multiple Interaction Negotiation

Multiple Interaction Negotiation (MIN) is initiated and terminated by confirmed services thus ensuring the agreement of both the VT-users and the service provider to the proposed action.

MIN is provided to enable the negotiation of a set of VTE-parameter values comprising a full-VTE in a series of steps (the intermediate steps do not necessarily form a complete or consistent full-VTE).

MIN negotiation is supported by four services:

a) INVITE: a VT-user invites the peer VT-user to propose values for VTE-parameters;
b) OFFER: a VT-user proposes values or value ranges to the peer VT-user; this may be in response to an INVITE or as a counter-offer to a previous OFFER;
c) ACCEPT: a VT-user accepts all or a subset of the VTE-parameter values proposed by the peer VT-user in an OFFER;
d) REJECT: a VT-user rejects VTE-parameter values proposed by the peer VT-user in an OFFER.

Using these services, independent sequences of MIN service primitives may be used to negotiate values for varying groupings of VTE-parameters. Valid MIN sequences and the rules for use of the MIN services are defined in 30.3.

NOTE – The stages of a MIN negotiation may be independent or may depend on the results of earlier stages.

When MIN is terminated, either

- the draft-VTE replaces the original current-VTE and is available for use, or
- the draft-VTE is discarded and the situation prior to the entry to MIN is restored; this may be with no full-VTE in existence.

Definitions of service phases associated with MIN are given in clause 27.

10.2 Negotiated Release functional unit

The Negotiated Release functional unit enables a VT-user to reject a request by the peer VT-user to release a VT-association and return to the phase of the VT service in effect before the release request. If this functional unit is not selected, a release request cannot be rejected.

10.3 Urgent Data functional unit

The Urgent Data functional unit provides a capability to allow a small amount of information to be conveyed from one VT-user to its peer in an urgent manner, possibly bypassing previous information exchanges. This functional unit is used to enhance support for the non-destructive interrupt facility, see 7.7, 20.1.4 (note 2) and B.16.2.

10.4 Break functional unit

The Break functional unit supports the destructive interrupt facility, see 7.7 and clause 34.

10.5 Enhanced Access-rules functional unit

The Enhanced Access-rules functional unit extends the access-rules available for control objects. In A-mode, the rule no-access is added. In S-mode, the rules WACI, WACA, WAVAR & WACI, WAVAR & WACA and no-access are added, see table 1. If this functional unit is not selected the current-VTE cannot contain COs which use these additional values of CO-access.

Table 1 lists the available access-rules in both modes of operation with and without this functional unit, see clause 9.

10.6 Structured Control Objects functional unit

The Structured Control Objects functional unit allows a parametrically defined control object to have more than one data element where the category of each data element need not be identical. This functional unit allows independent updating of the individual data elements. It also allows a control object defined non-parametrically (either in this International Standard or in a VTE-profile or as a registered control object) to be partially updated in a way defined specifically for that control object.

If this functional unit is not selected, the current-VTE cannot contain control objects requiring the above capability.

10.7 Blocks functional unit

The Blocks functional unit provides block structuring capability for a display object in a VTE. Actual use of the capability is optional, being selected for a display object by a VTE-parameter. If this functional unit is not selected the current-VTE cannot contain display objects with this capability.

10.8 Fields functional unit

The Fields functional unit provides field structuring capability for a display object in a VTE. Actual use of the capability is optional, being selected for a display object by a VTE-parameter. If this functional unit is not selected the current-VTE cannot contain display objects with this capability.

10.9 Reference Information Objects functional unit

The Reference Information Objects functional unit allows this particular form of control object to be used in a VTE. If this functional unit is not selected, the current-VTE cannot contain Reference Information Objects.

10.10 Ripple functional unit

The Ripple functional unit provides insertion, deletion and copy operations for a display object. These operations are available in two forms, either using basic addressing or using logical addressing. If this functional unit is selected, the Structured Control Objects functional unit must also be selected (so that the RMCO, 20.3.9, can be used).

NOTE – These functions allow simple text editing to be performed with much lower communications overhead than if the functional unit were not selected. Without this facility, editing of text with insertion and deletion would require the transmission of large parts of the Display Object between the peer VT-users. With this facility, only information relating to the actual insertion and deletion operations need be transmitted. Considerable communication savings are therefore possible.

10.11 Exceptions functional unit

The Exceptions functional unit provides mechanisms by which non-fatal exception conditions may be reported by the VT-service-provider to both VT-users. A facility is provided to ensure that the integrity of the VT-context is maintained.

NOTE – Without this facility, the VT-Service-provider normally aborts the VT-association if any exception condition is discovered. With this facility, exception conditions may be classified as either fatal or non-fatal. Fatal exception conditions will still cause the VT-service-provider to issue a VT-P-ABORT primitive.

10.12 Context Retention functional unit

The Context Retention functional unit allows the retention of the information stored in selected VT objects (DOs and COs) to be retained between successive VTEs within the life time of a VT-association.

The Context Retention functional unit may be selected only if the Switch Profile Negotiation functional unit is selected.

NOTE – This functional unit will allow, for instance, the contents of a display object to be retained when a new control object is brought into existence through negotiation. Without this functional unit, such contents would be lost.

11 VT Environment Profiles (VTE-profiles)

A VTE-profile is a set of VTE-parameters with predefined values for some or all of these VTE-parameters. A VTE-profile can have a number of VTE-profile arguments which are used to provide values for any VTE-parameters in the VTE-profile which are not given pre-defined values.

Other options can be defined in the VTE-profile as being subject to special VTE-profile arguments, but a VTE-profile definition cannot extend the provisions of this International Standard (e.g., new VTE-parameters not in this International Standard cannot be defined in this way).

In order to construct a full-VTE from such a VTE-profile, explicit values must be negotiated for all VTE-profile arguments (unless the VTE-profile itself defines a default value).

VTE-profiles are used to define control object and device object semantics.

Two default VTE-profiles are defined for this service. The default VTE-profiles, one for S-mode and the other for A-mode, are defined in annex A together with a description of the notation used for defining VTE-profiles and VTE-profile arguments.

NOTES

1 VTE-profiles provide an efficient method for defining the values of a number of VTE-parameters simultaneously. They also provide a means of promoting interoperability by registering well known VTE-profiles which have been designed for a particular purpose.

2 ISO/IEC 9834-4 contains the registration procedures for the ISO Registration Authority for VTE-profiles. Widely applicable VTE-profiles will be registered and maintained by that Registration Authority. The purpose of this registry is to promote interoperability and provide ease of reference.

Other VTE-profiles may be defined for use with the VT Service. These VTE-profiles may be registered with other registration authorities as appropriate for their level of visibility. In addition, private VTE-profiles may be defined and used outside the scope of ISO.

12 The VTE Model

Clauses 12 to 24 inclusive define the VTE Model in terms of which the Virtual Terminal Service is defined.

VT-users communicate by means of a shared conceptual communication area (CCA) containing a number of abstract objects and object type definitions. Information exchange is modelled by one VT-user updating the contents of the CCA and the changed state of the CCA then being made accessible to the peer VT-user. The VT-users update the CCA via the services provided by the service provider (see figure 2).

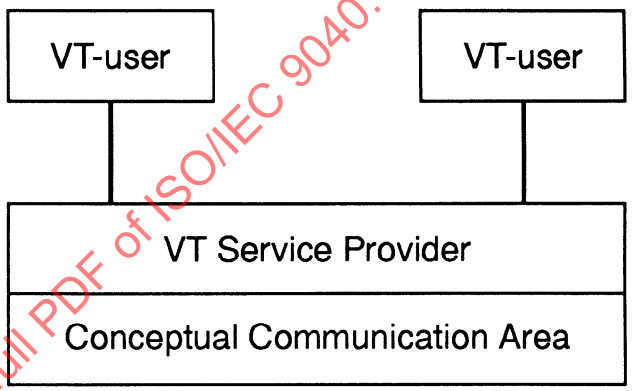


Figure 2 - Service model

The CCA, containing the abstract objects used for describing the VT service, has no physical existence.

The CCA contains the following components:

a) a CDS containing one or two display objects;

b) a CSS containing zero or more control objects which may be used not only for device control but also for signalling and status information and for any other purposes agreed outside the scope of the standard;

c) an ACS which records which VT-user, if any, owns the WAVAR access-right;

d) zero, one or more device objects each of which represents a mapping between a display object and a real device and provides VTE-parameters which enable some control over this mapping. A device object is linked to one display object and to one or more control objects; clause 16 describes the concept in more detail;

e) a DSD that contains the object type definitions for the display objects, device objects, control objects (see clauses 13, 14 and 16) and other negotiated VTE-parameters which form the principal part of the current-VTE definition.

Figure 3 illustrates these components of the CCA and their relationships (i.e., linking together) and also indicates how display objects and control objects can be mapped onto real devices via related device objects.

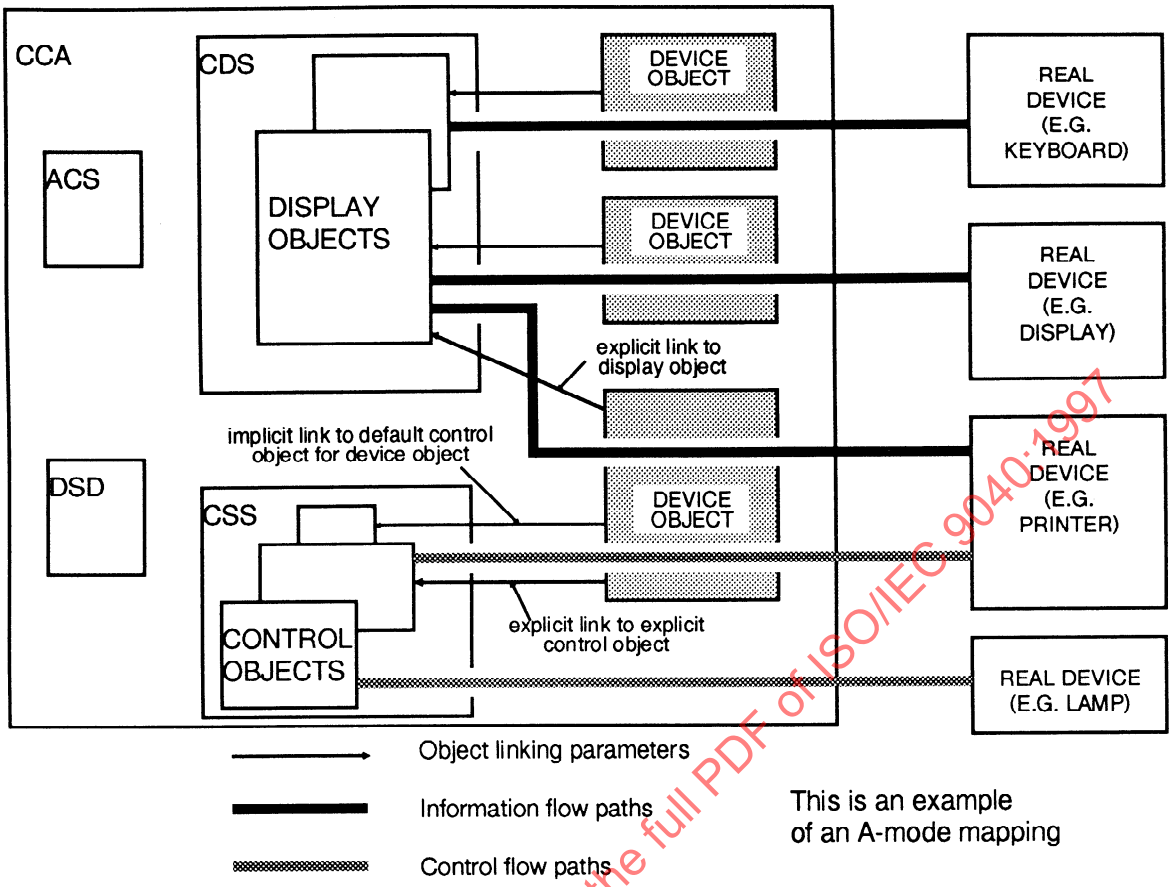


Figure 3 - CCA components, linking and mapping

The contents of the DSD are initialised to correspond to the current-VTE at VT-association establishment time and may only be changed subsequently by negotiation. The contents of the remaining sub-areas are open to change when the VT-association is in existence and negotiation is not in progress.

NOTES

1 The model described thus far is symmetrical. In actual use, however, the communication facilities provided by this model may be used in an asymmetrical manner. The method for assigning asymmetrical roles in practice uses the access-rules WACI and WACA, which are available in S-mode as well as A-mode when the Enhanced Access-rules functional unit is in use, see clauses 8 and 9.

Assignment of one of these access-rules to a RIO (clause 15) or any control object (clause 20 and clause 14) can have implicit semantics relating to the 'ownership' or 'location' of the object.

An asymmetry to the VT-users' view of the use of device objects can be assigned by means of the *device-default-CO-access* VTE-parameter (clause 23) but this asymmetry of device object use has no meaning to the VT service provider.

2 Clauses 25 onwards define the communication facilities and the primitives which VT-users use to modify the CCA. This International Standard does not preclude local mechanisms or "macro" operations which combine primitives at a local interface.

13 Display Objects

This International Standard does not define the semantics associated with the structure of a Display Object (DO), nor the relationship between this structure and the real device. Such semantics, i.e., the meaning of the update operations which are performed on the display object structure, depends on the VTE-profile used (see clause B.11).

The Kernel functional unit of the Basic Class Virtual Terminal Service defines a basic structure for the display object. This basic structure can be significantly extended if use of the Blocks or Fields functional units is selected for a particular DO, see 10.7, 10.8, 13.1.2 and 13.1.3.

Blocks are deemed to be in use for a particular DO if VTE-parameter *block-definition-capability* is "yes" for that DO. Fields are deemed to be in use for a particular DO if VTE-parameter *field-definition-capability* is "yes" for that DO (see 18.1).

13.1 Structure

A DO has the following components:

a) a one, two or three dimensional character array of array elements, each capable of holding one character box graphic element from a repertoire of such elements (see 13.2 and 18.2.4);

b) a display pointer, see 13.1.1.1;

c) a set of modal attribute values, see 13.1.1.2;

d) a set of global attribute values, see 13.1.1.3;

e) possibly, a logical pointer, see 13.1.3.3;

f) possibly, sets of field modal attribute values, see 13.1.3.2.

13.1.1 Character array

Each array dimension has a set of integer coordinate values 1...n so that the lower bound is 1. An upper bound may be defined or the array dimension may be declared unbounded (no limit to the value of n). There are no restrictions on combinations of bounded and unbounded dimensions.

The dimensions are named X, Y, and Z. X is the lowest order dimension. A set of array elements identified by a contiguous set of x-coordinate values 1...n (where n is the bound of X if X is bounded) at a particular value of y is known as an **X-array**. Y is the next higher order dimension. A set of contiguous y-coordinate values 1...n (where n is the bound of Y if Y is bounded) at a particular value of z identifies a set of X-arrays known as a **Y-array**. Z is the highest order dimension. A set of contiguous z-coordinate values 1...n (where n is the bound of Z if Z is bounded) identifies a set of Y-arrays known as the **Z- array** (see also clause B.11).

The dimensions determine the ordering of all the addresses of the display object in that address (x,y,z) is less than (precedes) (x',y',z') precisely if

either $z < z'$
or $z = z'$ and $y < y'$
or $z = z'$ and $y = y'$ and $x < x'$.

This ordering rule applies to the operations defined in clause 19.

13.1.1.1 Display Pointer

The display pointer is either a primitive display pointer or an extended display pointer. The primitive version applies when blocks are not in use. The extended version applies when blocks are in use. The **primitive display pointer** consists of a set of (one, two or three) coordinate values which identifies a particular array element or can identify a position immediately after a bounded X-array where there is no array element (see 19.1.1.1). The **extended display pointer** consists of a set of coordinate values (p,q,b,z), where q and z are only used if the y and z dimensions respectively are defined. It (usually) identifies an array element, as defined in 13.1.2. The initial value for either pointer when a VTE is first brought into use (initialized) is 1 for all applicable coordinate values.

13.1.1.2 Modal Attributes

The set of modal attributes consists of a value for each of the secondary attributes (see 13.2). Each value is either "null" or is an explicit value for the attribute from the set of values negotiated for this attribute. These values are used as defined in clause 19. The initial values, when a VTE is first brought into use (initialized), are "null" for all attributes.

13.1.1.3 Global Attributes

The set of global attributes consists of a value for each of the secondary attributes (see 13.2) except character repertoire. Each value is either "null" or is an explicit value for the attribute from the set of values negotiated for this attribute. These values are used as defined in clause 19. The initial values when a VTE is first brought into use (initialized) are "null" for all attributes.

13.1.2 Block definition

A **block** is a rectangular sub-area of a Y-array specific to that Y-array. It has an origin designated by an (x,y) coordinate value and a width in X and a depth in Y denoted as Dx, Dy respectively. Any edge of a block can coincide with any edge of the Y-array but cannot go outside the bounds of X and Y. Dx and Dy must be positive and non-zero. When the block-definition-capability is used, the X dimension and, if defined, the Y dimension must be bounded.

Figure 4 illustrates the block concept.

If the Y-dimension is not defined for the display object, the Y-array and all blocks are implicitly one X-array deep, and all references to Y dimension must be amended accordingly.

A block establishes an addressing sub-domain of the Y-array which contains some subset of the array elements of the Y-array. Within the block the contained array elements are addressed using p and q coordinate values which are interpreted as relative to the origin of the block. The p and q coordinates are subject to the addressing constraints negotiated for the X and Y dimensions respectively of the unstructured DO.

The subset of an X-array which is contained within a block is termed an X-subarray or P-array.

A number of blocks can be defined in each Y-array. The maximum number is a negotiable parameter of the DO and can be unbounded. The block structure on a Y-array is independent of that on any other Y-array. The blocks of a Y-array are addressed by a pseudo-dimension known as the B dimension, with b used to denote the coordinate value.

The address of a block is a value (b,z), the address of a X-subarray (P-array) in a block is (q,b,z) and of an array element within a block is (p,q,b,z), where q and z are used only if Y and Z dimensions respectively are defined. The address (p,q,b,z) identifies the array element (x,y,z) where

$$x = X_o + p - 1$$

$$y = Y_o + q - 1$$

and (X_o, Y_o) is the origin of the block (b,z).

Extended addresses, i.e., including the block coordinate, are ordered by an extension of the ordering algorithm in 13.1.1. Address (p,q,b,z) is less than (precedes) (p',q',b',z') precisely if

either $z < z'$
or $z = z'$ and $b < b'$
or $z = z'$ and $b = b'$ and $q < q'$
or $z = z'$ and $b = b'$ and $q = q'$ and $p < p'$.

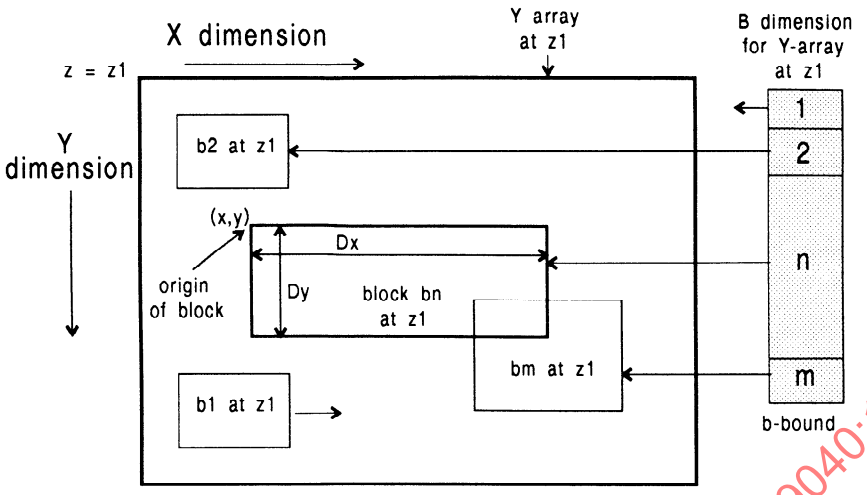


Figure 4 - Illustration of Block concept

Blocks can overlap. Thus, an array element of the Y-array can be addressed and updated by more than one block addressing mechanism, but has single values for primary and secondary attributes. See also 13.1.3.

If blocks are in use for a DO, extended addressing always applies and an array element cannot be addressed (or updated) using extended addressing unless it is contained within at least one block. See also 13.1.3.3 and 19.5.

Operations are defined in 19.1.2 and 19.4.1.5 for creating and deleting blocks and for other update operations using blocks.

NOTES

1 The creation or deletion of one or more blocks does not affect the content of any array element which is always that resulting from the latest update made by any mechanism, see 19.4.1.5.

2 In contrast to fields, see 13.1.3, no explicit storage is included in the model for block definition information.

13.1.3 Field definition

A **field** is a logical sub-area of a Y-array; it is an addressing domain with one-dimensional addressing which is independent of the actual form of the field in terms of its structure on the underlying display object. When the field-definition-capability is used, the X dimension and, if defined, the Y dimension must be bounded. Additional attribute capabilities are available with fields.

A single field is constructed of a sequence of one or more non-overlapping rectangular sub-areas of a particular Y-array known as field-elements; these are defined in a similar way to blocks but are formally distinct from blocks (they do not appear on the B dimension for the Y-array). The logical ordering of the field-elements is determined by the field definition not the position on the Y-array. See also 13.1.3.1.

A field can be of any shape within the bounds of the Y-array but may not overlap another field. A field may overlap one or more blocks. Thus an array element of the Y-array can be addressed and updated by one field and by one or more block addressing mechanisms, but has single values for primary and secondary attributes.

A number of fields can be created on a Y-array. The maximum number is a negotiable parameter for the DO and can be unbounded. The field structure on a Y-array is independent of that on any other Y-array. The fields of a Y-array are addressed by a pseudo-dimension known as the F dimension, with f used to denote the coordinate value. Each value of f can potentially address one complete field, however many field-elements it contains (but at any particular time the field may be "non-extant", see 20.3.3.5).

The 'logical' addressing of array elements within a field is denoted by the pseudo-dimension K with k used to denote a coordinate value on this. See also 13.1.3.1.

Figure 5 ilustrates the field concept.

Fields are addressed using the logical pointer, see 13.1.3.3. The address of a field is a value (f,z) and the address of an array element within a field is (k,f,z), where z is used only if the Z dimension is defined. The relative k coordinate is interpreted such that value 1 (the minimum) addresses the origin position of the first field-element of the field in the field definition. The maximum value for k is the total number of array elements contained in all the field-elements in the field. (A field cannot be "unbounded".)

NOTE 1 – The z coordinate of the logical pointer is independent of the display pointer for the underlying display object. Thus, when switching between logical addressing and either block addressing or addressing the underlying display object, the values of the respective pointers are preserved.

Logical addresses are ordered as follows: address (k,f,z) is less than (precedes) (k',f',z') precisely if

either $z < z'$
or $z = z'$ and $f < f'$
or $z = z'$ and $f = f'$ and $k < k'$.

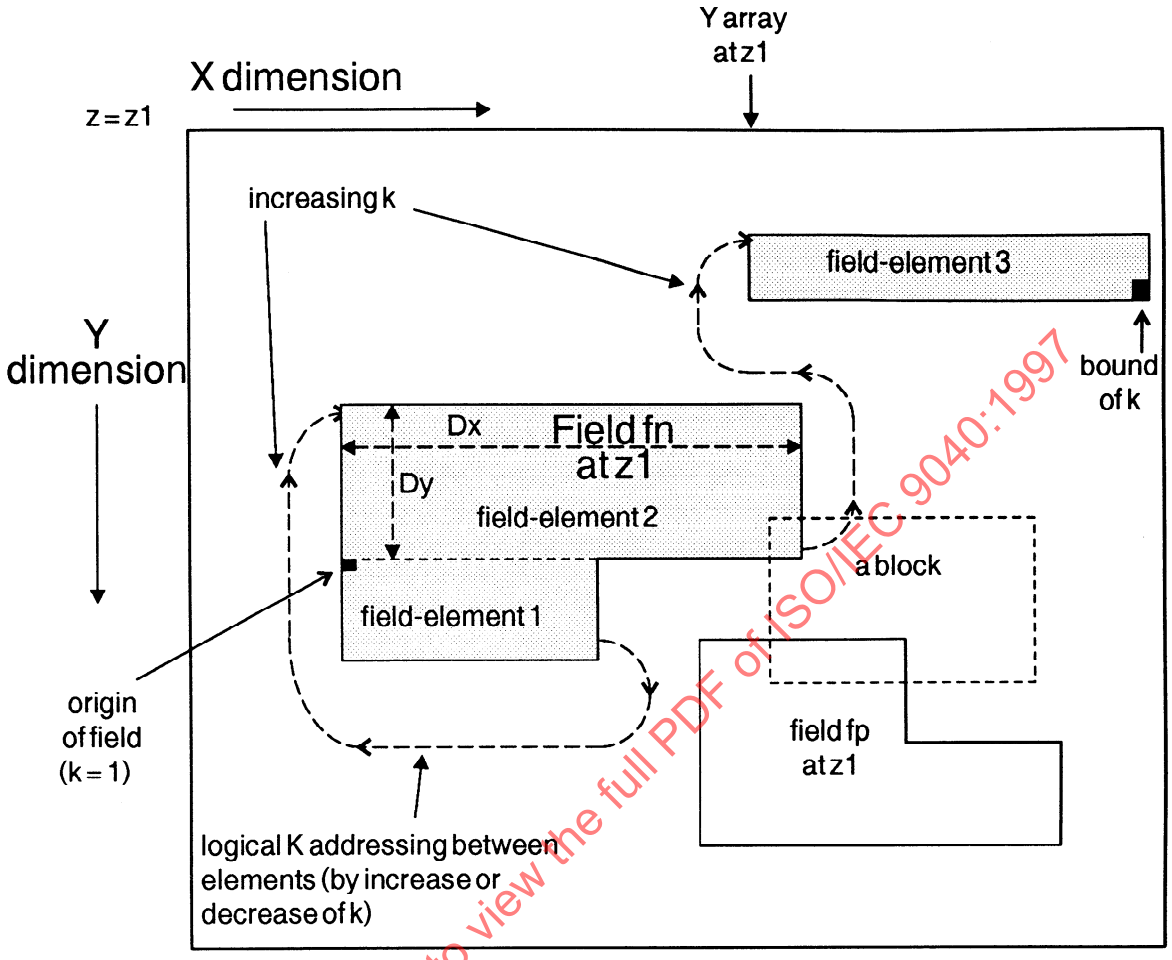


Figure 5 - Illustration of Field concept

NOTE 2 – The presence of explicit navigation path links, see 20.3.3.4, has no effect on the ordering as defined by the above algorithm and as used by operations which refer to this ordering.

The relationship between the k coordinate values and the (x,y) coordinates of the array elements contained in a field is defined in 13.1.3.1.

Field definitions are held in a FDCO associated with the DO, see 14.2 and 20.3.3. Fields are Created, Modified or Deleted by update operations on the FDCO, see 20.3.3.5. A particular (z,f) value may or may not have an associated defined field at any particular time.

Operations are defined in clause 19 for update operations using fields.

In addition to the normal access control applied by the display object access-rule, further rules for the control of updates by the terminal VT-user can be applied using the FEICO and FEPCO types of control object, see 19.5 and clause 20.

When field-definition-capability is selected for a DO, update access to the DO other than by the logical pointer (13.1.3.3) is subject to the VTE-parameter *access-outside-fields* (18.2.2); see also 19.5.

13.1.3.1 Logical addressing by K dimension

The following defines the structure of a field and defines the relationship of the logical addressing of the contained array elements by the K dimension to the (x,y) coordinates on the underlying Y-array (display pointer addressing).

The field-elements of a field are similar in shape to blocks, having a width in the X-dimension and a depth in the Y-dimension, see 13.1.3. For the purpose only of defining the K-dimension, a pseudo-dimension referred to as the S-dimension is introduced. The field-elements are arranged on this in the order in which they are defined in the field extent in the FDR (20.3.3.3), and are considered to have S coordinate values from 1 to some value S_m determined by the number of field-elements (may be 1). S coordinate values do not appear in any pointer and no means of explicitly changing s coordinate value is provided.

The K-dimension in effect links through all the field-elements of one field in increasing S coordinate value. Thus, k=1 corresponds to (1,1,s=1) and increasing k values move along the first x-subarray to its bound (x_m,1,s=1). Unless this is the last X-subarray in the field-element, an automatic next-X-subarrray operation, to (1,y:=y+1,s=1), occurs. Decreasing k

values will cause the reverse move. At the bound (x_m,y_m,s) of the last x-subarray in a field-element (not being the last field-element of the field), an automatic 'next-field-element' operation occurs, i.e., to (1,1,s:=s+1). Decreasing k values move from the origin (1,1,s) of one field-element (not being the first) with an automatic 'previous-field-element' operation to (x_m,y_m,s:=s-1), i.e., to the maximum values of x and y for field-element s-1. The bound of K is when x and y reach their maximum values in the last field-element S_m. See also B.18.2.

13.1.3.2 Modal Attributes for Fields

When field-definition-capability is selected for a DO, there is a capability for associating with each field a set of secondary attribute values known as the **field modal attributes**. Each set consists of a value for each of the secondary attributes (see 13.2). Each value is either "null" or is an explicit value from the set of values negotiated for this attribute. These field modal attribute values are set or used by certain operations on fields as defined in clause 19.

NOTE – Although field modal attributes are not formally part of the FDRs which otherwise define the fields, it is implicit that a set of field modal attributes can be stored for every FDR which can be in an "active" or "inactive" state, see 20.3.3.

13.1.3.3 Logical Pointer

When fields are in use for a DO, an additional form of pointer is brought into use, the **logical pointer**. This has the form (k,f,z) where z is present only if Z dimension is defined. The initial value for the pointer when a VTE is first brought into use (initialized) is 1 for all applicable coordinate values.

The display pointer (in primitive or extended form) and logical pointer are independent of each other and a change to one does not affect the value of the components of the other. Thus, when changing between logical addressing and either block addressing or addressing the underlying display object, the values of the respective pointers are preserved.

NOTES

1 When fields are in use for a DO, an array element which is not in the address domain of any field (when mapped onto underlying physical coordinates) cannot be addressed by the logical pointer and thus cannot be updated by any logical operation.

2 An array element for which the condition in note 1 above is satisfied and which is not within a block cannot be addressed or updated.

13.2 Attributes

At any given time, an array element is either empty (i.e., it has no primary attribute value currently assigned to it) or it contains a **primary attribute** value selecting a character-box graphic element from a repertoire of such elements.

Each array element also has a number of **secondary attributes**:

- character-repertoire;
- font;
- emphasis;
- foreground-colour;
- background-colour.

The last four are referred to as the **rendition attributes**.

Each rendition attribute is independent of other rendition attributes and can have multiple values negotiated for its use.

The rendition attribute *font* is subservient to character-repertoire in that a separate set of font values for each repertoire can be negotiated with semantics specific to that repertoire.

The rendition attribute *emphasis* may have a substructure; the individual subattributes may be individually updatable.

If the primary attribute in an array element has a value assigned, then all secondary attributes necessarily have values assigned. If the primary attribute has no value assigned, each rendition attribute may or may not have a value, independently, and the character-repertoire secondary attribute has no value assigned.

Each array element, with its content of primary and secondary attributes, is self-contained and completely independent of all other array elements.

NOTES

1 The interpretation of any empty array element and the effect of any rendition attribute values, and in particular its physical representation on any real device, e.g. a display or printer, are not defined in this International Standard.

2 This International Standard does not further define the form of the emphasis attribute or whether any component subattributes are individually updatable. It includes VTE-parameters to enable a VTE-profile to define a form for this attribute and provide profile arguments for precise assignment in a VTE; see 18.2.6 and clause 23.

3 The additional facilities described in 13.1 do not alter the property that each array element of the display object has a single value for the primary attribute and each secondary attribute independently of the values in any other array elements (this value may be undefined). They do provide additional ways in which these values can be set.

14 Control objects

Control objects (COs) enable VT-users to handle control information related to virtual terminal functions and to real devices. This International Standard defines VTE-parameters determining the syntax of control objects, that is, their range of values and permitted update operations, see 20.1.

The semantics of a control object, i.e., the interpretation of update operations in relation to real devices, is determined by the CO-type-identifier VTE-parameter, see 20.1.1. This specifies the semantics of the CO as being defined by one of the following:

a) within this International Standard;

b) in a VTE-profile;

c) as part of a registered control object;

d) by means known to both VT-users outside the scope of this International Standard.

When a VT-association is established, the control objects defined in the VTE-profile selected are the control objects available in the initial VTE. Subsequent negotiation may

e) replace the current VTE (thus making a new set of control objects, as defined in the new VTE-profile, available),

f) add new control objects to the current VTE, or

g) alter the characteristics of existing control objects.

A default control object is implicitly defined when each device object is defined. This default control object contains eight booleans and may be defined as having a trigger mechanism, see clauses 16 and B.4.

Each control object is assigned an unambiguous name within the scope of the VTE. This name is used to identify the control object when its information field is to be updated or its VTE-parameters are to be negotiated. A default control object is referenced using the name of the device object with which it is associated.

A control object may be linked to a device object (see clauses 16 and 23).

14.1 CO structure facilities and related restrictions

Two functional units are relevant to general control object capabilities, the Enhanced Access-rules functional unit and the Structured Control Objects functional unit.

14.1.1 Access-rules

A control object has an access-rule determining which VT-user may update it and when, see clause 9. The Enhanced Access-rules functional unit determines the available set of access-rules which can be assigned to control objects as defined in table 1.

A control object also has an update priority, see 20.1.4 and 24.5. The Enhanced Access-rules functional unit determines the values of VTE-parameter *CO-priority* which can be assigned, see 20.1.4 and table 11.

If the *CO-priority* value is "normal", the control object may also have the "trigger" characteristic. An update to a CO with the trigger characteristic causes delivery of queued updates and (in S-mode) transfers the WAVAR access-right to the peer VT-user.

14.1.2 CO information structure

The Structured Control Objects functional unit determines the capability available for defining structure in the information content of a CO.

In general, the structure of the information field of a control object may be defined either in terms of VTE-parameters or by reference to a separate definition which then accompanies the definition of the information field semantics. This separate definition can be contained within this International Standard or can be external to it, for example, in an international register.

When the information structure is defined parametrically it comprises a number of data elements; the number and the type and size of each data element are VTE-parameter values and thus can be changed by negotiation. Data elements thus defined may be individually updated without simultaneously updating other data elements of the same object.

The number of data elements can exceed one only if the Structured Control Objects functional unit is selected for the VT-association.

When the information structure is not defined parametrically changes to the elements of the structure cannot be negotiated; the structure definition may allow individual updating of elements. The information structure of the CO must be defined parametrically if the Structured Control Objects functional unit is not selected.

14.2 Standard control objects for fields and controlled data entry

This International Standard defines a number of standard types of control object which are used with the Fields functional unit. These are listed below with a brief overview of their use. Use of these COs also requires the Structured Control Objects functional unit to be selected, see clause 10. Clause 20 defines the VTE-parameters for these control objects and gives further details of their use. Annex B gives further explanatory material.

a) **Field Definition Control Object FDCO**: mandatory if field-definition-capability is to be used for a display object. Fields are defined by Field Definition Records (FDRs) in the FDCO. FDRs can also link to records in FEICO and FEPCO, see below, to apply conditions for controlled data entry to the fields. The access-rule for the FDCO determines the role-assignment, if any, known to the service, see clause 12 and 19.5.2;

b) **Field Entry Instructions Control Object FEICO**: optional, but useful only if a FDCO is present, see a); contains Field Entry Instructions (FEIs) which can be linked to a field by the FDR of the field;

c) **Field Entry Pilots Control Object FEPCO**: mandatory if a FEICO is present, see b), otherwise optional, but useful only if a FDCO is present, see a); contains data entry pilots each consisting of a Field Entry Event (FEE), Field Entry Conditions (FECs) and a sequence of Field Entry Reactions (FERs) and which can be linked to a field by the FDR of the field;

d) **Context Control Object CCO**: optional but useful only if a FDCO is present, see a); it enables an initial starting point for data entry to be indicated and similarly the point at which termination of data entry occurred, with the reason for termination (if a FEPCO is in use);

e) **Transmission Policy Control Object TPCO**: optional but useful only if a FDCO is present, see a); contains general instructions relating to the notification of field updates which are used as the default for all fields but which for any field can be over-ridden by its FDR.

For a display object for which field-definition-capability is selected, exactly one FDCO must be defined, at most one CCO and TPCO can be defined, and one or more FEICOs and FEPCOs can be defined; FEPCOs are not directly associated

with FEICOs but at east one FEPCO must be present if any FEICOs are present. The various COs for a display object should be linked to it as recommended in 23.5; B.18.14 illustrates this. B.18.4 and B.18.5 give further explanation of the use of data entry COs.

14.3 Standard control object for dynamic termination conditions

This International Standard defines a standard type of CO, the **Termination Conditions Control Object TCCO**, which can hold dynamic termination conditions. It is available in either mode of operation. Its use requires the Structured Control Objects functional unit to be selected. A number of instances of the type can be used in a VTE, each with a VTE-unique *CO-name* value. The termination conditions in such a CO can be updated at any time when a VT-user has update access to it. If one or more device objects link to such a CO then all the termination VTE-parameters in these device objects become ineffective. However, the TCCO is overridden for a device object if that device object is linked to a FDCO, see 14.2 and 23.5. See clause 20 for further definition.

14.4 Standard control object for notifying termination

This International Standard defines a standard type of CO, the **Termination Control Object TCO**, which is used to notify the Application VT-user of the reason input was terminated and made available by the Terminal VT-user. It is available in either mode of operation and does not require the Structured Control Objects functional unit. It may be used in conjunction with termination events specified in either the termination VTE-parameters in a device object or termination events specified in a TCCO. See clause 20 for further definition.

14.5 Standard control object for echo control

This International Standard defines a standard type of control object, the **Echo Control Object ECO**, which may be used in A-mode to notify the remote VT-user whether echoing is on or off (echoing is the responsibility of the Terminal VT-user and this control object only provides an indication of whether the Application VT-user wants the Terminal VT-user to echo or not). This control object does not require the Structured Control Objects functional unit. See clause 20 for further definition.

14.6 Standard control object for ripple mode editing

This International Standard defines a standard type of control object, the **Ripple Mode Control Object RMCO** which is used as part of the ripple mode editing facility made available with the Ripple functional unit. For a display object for which ripple-capability is selected, exactly one RMCO must be defined. Use of this CO type requires the Structured Control Objects functional unit to be selected. Clause 20.3.9 defines the VTE-parameters for this CO and gives further details of its use.

The following information is held in a RMCO:

a) a boolean switch to allow a VT-user to alternate between insertion and overwrite operation of DO updates;

b) the primary and secondary attributes to be applied to empty space resulting from ripple operations;

c) a control which governs the action to be taken in overflow situations at the end of an x-array or a y-array;

d) a control which governs the action to be taken in overflow situations when using logical addressing;

e) a control to determine whether insert and delete array operations are to be before or after the display pointer.

15 Reference Information Objects

A Reference Information Object (RIO) is a specific type of structured control object which contains object update information for reference while a VTE is in use. The nature of this CO type is defined in this clause. Later clauses define the VTE-parameter values for this object type and operations on such objects; these are an extension to the set of operations defined for other COs defined in this International Standard.

15.1 Structure

A RIO is a container for update information that may be referenced during VT operation. It is structured into a number of smaller accessible units called RIO-records, each designed to contain an independently addressable element of such information.

A RIO-record may contain any valid content for a VT-DATA request, i.e., an arbitrary sequence of updates to display objects and control objects (including RIO updates) and execute- or call- record operations (see 22.4.1 and 22.4.2) on the same or a different RIO; nesting is permitted but loops must not occur. A RIO-record can exist but be empty.

A RIO has an access-rule which is agreed as part of the definition of a RIO in a VTE. It conforms to the general provisions for control objects.

15.2 Generation and use of RIOs

One or more RIOs can be included in a VTE only if the Reference Information Objects functional unit is selected. This functional unit is independent of any other VT functional unit.

VTE-parameters to define a RIO in a VTE are defined in clause 21. They conform to the general CO VTE-parameters in clause 20.

The initial content of a RIO can be defined in a Profile, called into a profile from a register of predefined RIOs, possibly by a profile argument, or can be called directly from the register into a VTE during negotiation. Initial content of a RIO derived in this way is implicit in its definition and cannot be negotiated. The initial content can be empty.

The initial contents of a RIO can be modified or new contents added (using VT-DATA) only if an access-rule other than no-access is assigned to the RIO; the definition of the RIO may forbid or restrict such an assignment.

Operations to update a RIO and to make use of its contents are defined in clause 22; these are extensions to the update operations as defined for other control objects.

NOTE – a RIO-record can also be used as a copy buffer, see 19.1.4.10 and 19.4.2.5; such use is independent of the usage described above.

16 Device objects

Device objects provide VT-users with a mechanism for specifying certain real device related characteristics to aid in mapping information contained in display objects to/from real devices. Device objects also serve to relate devices to display objects and any control objects which may have been selected as part of a VTE (see clause 14) to assist in the operation of the device.

Device objects are selected in the initial VTE when establishing a VT-association when they exist in the selected VTE-profile.

Subsequent negotiation may cause a different set of device objects to be selected if a new VTE-profile is used. Negotiation may also define new device objects using VTE-parameters (see clause 23) or alter the characteristics of existing ones.

A device object has a single name which is unambiguous within the scope of the VTE.

Device objects are not updated directly using the Data Transfer facility. The VTE-parameters of a device object are intended to influence the mapping between a display object and a real device.

A default control object is associated with each device object. This control object does not use the normal control object VTE-parameters (clause 20) but has pre-defined characteristics and certain specific VTE-parameters in the device object (see clause 23). For update purposes, this control object is referenced by the name of the device object.

17 VTE Parameters and directed graph

17.1 Directed graph of VTE-parameters

The VTE-parameters are structured as a directed graph as specified in figures 6 and 7. These figures also indicate those VTE-parameters which may have multiple occurrences.

For explanatory purposes, some sub-trees of the graph are shown with a "root" for which there is no actual parameter defined in this International Standard. These "phantom" roots are indicated by [...] and, of course, cannot be given values.

The individual parameters of the directed graph are defined in clauses 18, 20, 23 and 24 except for *mode*, *class* and *functional units* which are not VTE-parameters but are service parameters of the VT-ASSOCIATE service, i.e., *VT-mode*, *VT-class* and *VT-functional-units* respectively, (see 28.1.3).

NOTE – The directed graph does not show the dependency of certain VTE-parameters on VT functional units.

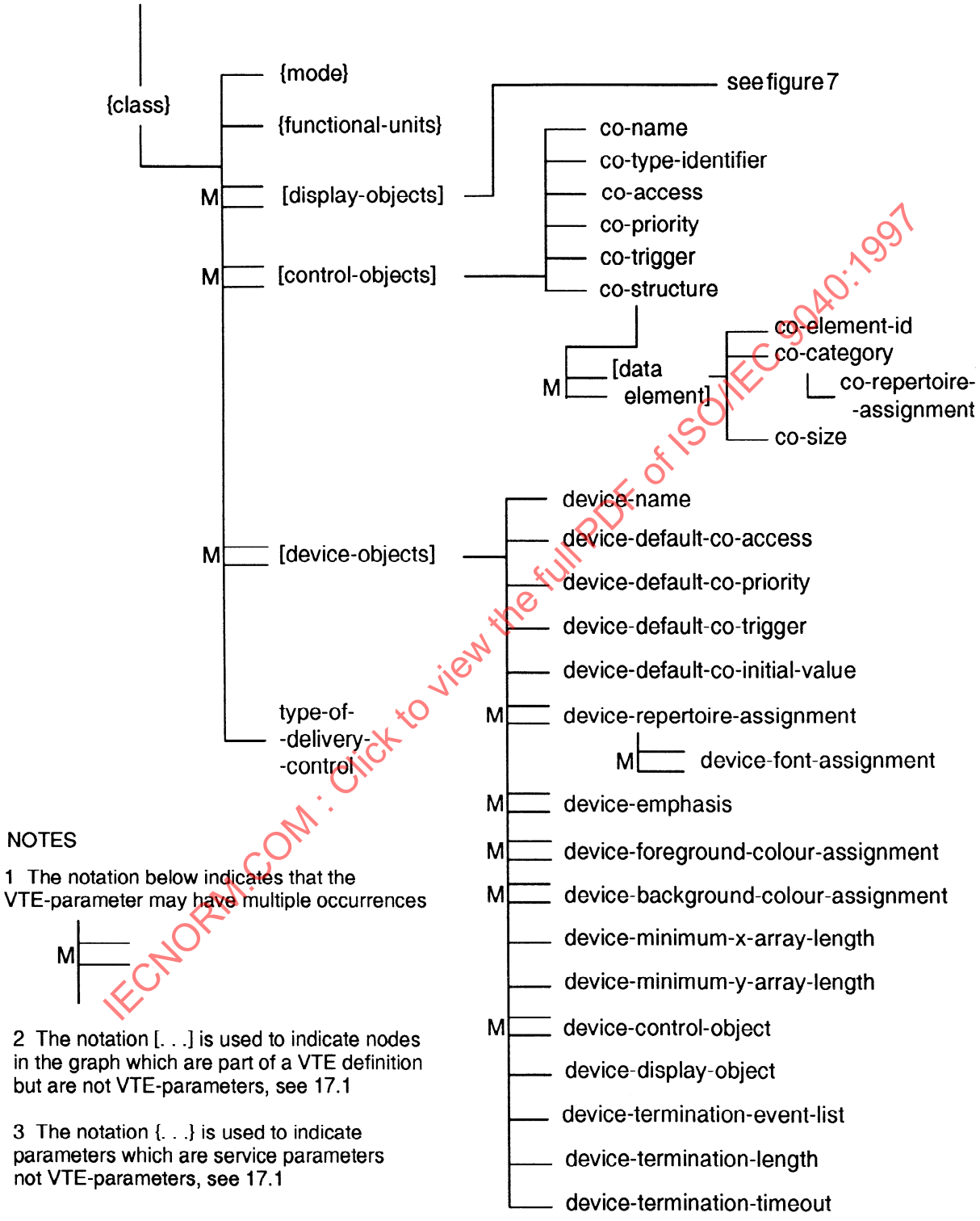
17.2 VTE consistency rules

A full-VTE is required to have values for VTE-parameters as follows:

a) a value for any VTE-parameter whose existence is implied by a "parent" VTE-parameter at a higher node in the graph or as defined in this International Standard; this value may be supplied as a default if one is defined;

b) values for multiple occurrence VTE-parameters whose existence is implied by a "parent" VTE-parameter at a higher node in the graph or as defined in this International Standard; a value may be supplied as a default if one is defined;

c) excluding values for VTE-parameters whose non-existence is implied by the value of a parameter at a higher order node in the graph or as defined in this International Standard.



NOTES

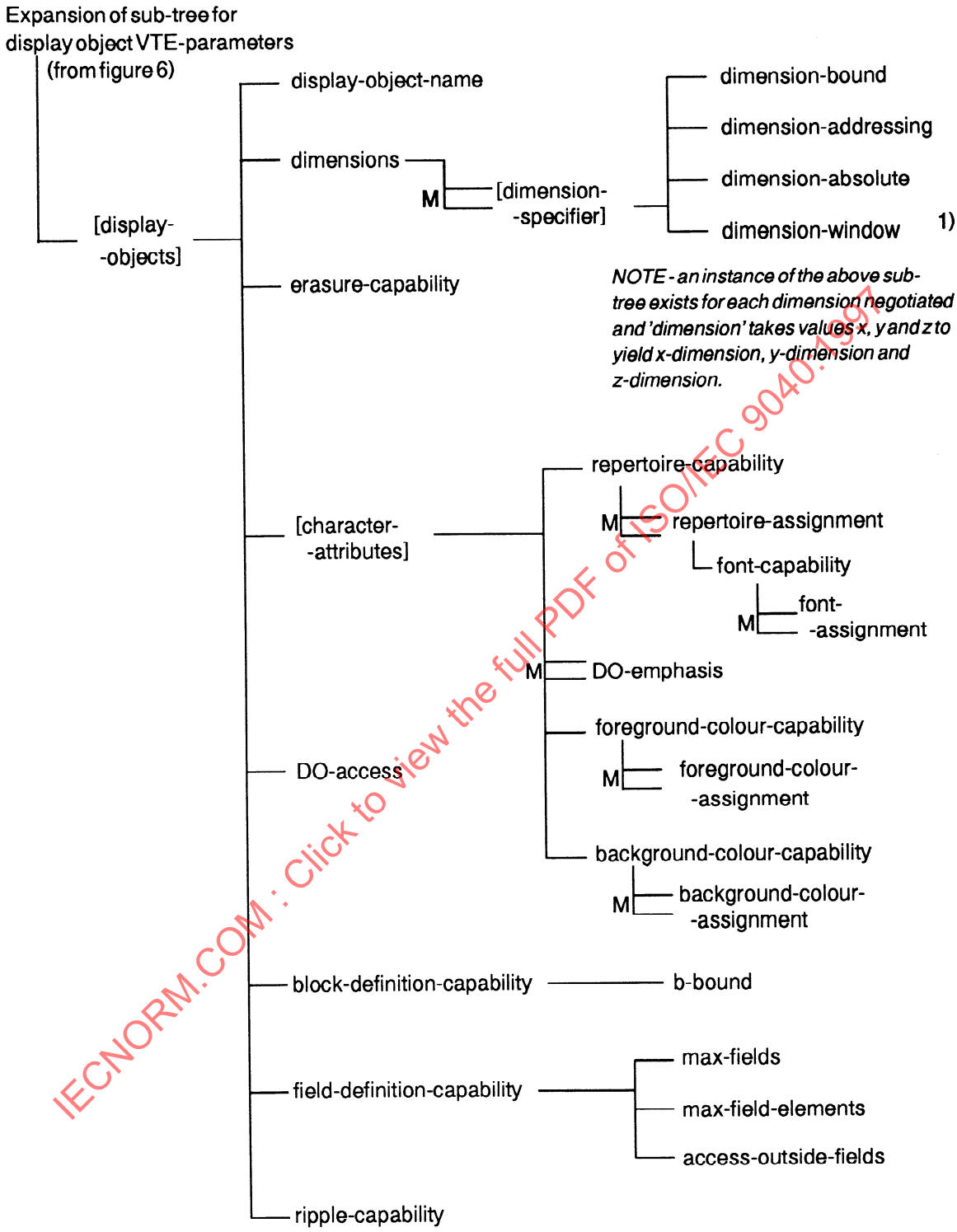
1 The notation below indicates that the VTE-parameter may have multiple occurrences

M

2 The notation [. . .] is used to indicate nodes in the graph which are part of a VTE definition but are not VTE-parameters, see 17.1

3 The notation {. . .} is used to indicate parameters which are service parameters not VTE-parameters, see 17.1

Figure 6 - Directed graph of VTE-parameters - part 1



NOTE - an instance of the above sub-tree exists for each dimension negotiated and 'dimension' takes values x, y and z to yield x-dimension, y-dimension and z-dimension.

1) When either block-definition-capability or field-definition-capability has value "yes for a DO the VTE-parameter dimension-window has an explicit value only for the Z dimension.

Figure 7 - Directed graph of VTE-parameters - part 2

18 Display Object VTE-parameters

This clause defines the VTE-parameters for display objects. These VTE-parameters have their values assigned through negotiation either at VT-association establishment time or by explicit negotiation subsequent to VT-association establishment. Their values determine the display characteristics of the VT service.

18.1 Primary VTE-parameters

The primary characteristics of a display object are defined by the VTE-parameters in table 2. These VTE-parameters establish the structure of the display object and the set of attributes that may be associated with each array element.

Table 2 - Primary VTE-parameters

Parameter	Value
display-object-name	character string of type ASN.1 PrintableString
DO-access	"WACI", "WACA", "WAVAR" (see clause 9)
dimensions	"one", "two", "three" (default = "two")
erasure-capability	"yes", "no" (default = "no")
block-definition-capability	"yes", "no" (default = "no")
field-definition-capability	"yes", "no" (default = "no")
repertoire-capability	1...N (default = 1)
foreground-colour-capability	1...N (default = 1)
background-colour-capability	1...N (default = 1)
ripple-capability	"yes", "no" (default = "no")

The VTE-parameter **erasure-capability** controls the availability of the ERASE operation (19.4.1.4) for the display object. The value "yes" indicates the erase operation is available; "no" indicates it is not available.

The VTE-parameter **block-definition-capability** may be used only if the Blocks functional unit has been selected.

The VTE-parameter **field-definition-capability** may be used only if the Fields functional unit has been selected.

The VTE-parameters **repertoire-capability, foreground-colour-capability** and **background-colour-capability** specify the number of attribute values that may be defined by the corresponding assignment VTE-parameters defined in 18.2.4 and 18.2.5.

NOTE – These VTE-parameters do not themselves define the individual attribute values. For example, the *foreground-colour-capability* value represents the number of different *foreground-colour-assignment* values which may be used.

The VTE-parameter **ripple-capablility** may be used only if the Ripple functional unit has been selected.

18.2 Secondary VTE-parameters

Certain of the VTE-parameters defined in this subclause and in 18.3 may have multiple occurrences in ordered lists. These lists are numbered starting from 1. During update operations, a value is selected by its position in the ordered list. The maximum number of occurrences in such an ordered list may be controlled by a higher order VTE-parameter of the "capability" type, see 18.1. A value "null" is available to act as a "placeholder" in an ordered list when an explicit value is not required. An ordered list may be truncated at the end.

In the case of the multiple occurrence VTE-parameters for repertoire and colour assignment in 18.2.4 and 18.2.5, the first entry in the ordered list is the **explicit modal default** value. Under certain circumstances, this value will be used by the TEXT, REPEAT-TEXT or ERASE operations, see 19.4.1.1, 19.4.1.2 and 19.4.1.4. In the case of the font attribute, the explicit modal default used by these operations is the first entry in the list of values of the *font-assignment* VTE-parameter (18.3) corresponding to the actual resulting value of the character-repertoire attribute in each array element on which the operation is performed. If the ordered list for one of these VTE-parameters is absent or if the first value is "null", the implicit default (defined in tables 4, 5 or 7) applies.

In the case of the emphasis attribute, the explicit modal default value is not defined in this International Standard, see B.17.

18.2.1 Block-definition-capability VTE-parameter

The following VTE-parameter is applicable only if VTE-parameter *block-definition-capability* takes value "yes".

b-bound: "unbounded",1...N (default = 1, single block).

It enables a limit to be set to the number of blocks which can be defined on any one Y-array.

If the value is an integer, i.e., the dimension is not "unbounded", it is invalid for an addressing operation (19.1.2.2) or a CREATE BLOCK or DELETE BLOCK operation (19.4.1.5) to use a value of b greater than this integer. A value of b less than 1 is always invalid.

18.2.2 Field-definition-capability VTE-parameters

The following set of VTE-parameters is applicable only if VTE-parameter *field-definition-capability* takes value "yes".

max-fields: "unbounded", 1...N (default = 1);
max-field-elements: "unbounded", 1...N (default = 1);
access-outside-fields: "allowed", "not allowed" (default = "allowed").

The VTE-parameter **max-fields** defines the bound of the F dimension, i.e., the maximum number of fields which can exist at any time on any one Y-array. If the value is an integer, i.e., the dimension is not "unbounded", it is invalid for a logical addressing operation (19.1.3) or a FDR update (20.3.3.2) to use a value of f greater than this integer. A value of f less than 1 is always invalid.

The VTE-parameter **max-field-elements** defines the maximum number of field-elements which can be used to construct any one field (i.e., the bound of the S pseudo-dimension, see 13.1.3.1). If the value is an integer, i.e., not "unbounded", it is invalid for field-extent in an FDR update (20.3.3.3) to contain more than this number of field-element-defining 4-tuples.

Either VTE-parameter can take the value "unbounded".

The VTE-parameter **access-outside-fields** value is significant if the VT-users are distinguished by the designations Application VT-user and Terminal VT-user, see clause 12, as follows. See also 19.5.

If the value is "not allowed" the Terminal VT-user is restricted to logical operations on the display object. If the value is "allowed" the Terminal VT-user is not restricted to logical operations on the display object.

Where there is no distinction between the VT-users, this VTE-parameter has no effect, see 19.5.

18.2.3 Addressing VTE-parameters

The VTE-parameters defining the display object addressing constraints are specified in table 3. They are always present if the corresponding dimension has been defined.

The VTE-parameter **x-bound** defines the upper bound of the x dimension. If the value is an integer, i.e., the dimension is not "unbounded", an explicit addressing operation is invalid if it attempts to set a value greater than this value plus one. An attempt to set a value less than one is always invalid.

Table 3 - Addressing VTE-parameters

Parameter	Value
x-bound	"unbounded", 1...N (default = "unbounded")
x-addressing	"no constraint", "higher only", "not permitted" (default = "higher only")
x-absolute	"yes", "no" (default = "no")
x-window	"unbounded", 0...N (default = 0 if x-bound = "unbounded" (i.e., no backward movement is permitted), otherwise default is value of x-bound (window aligned with dimension limits; see 19.1.1.3)
y-bound y-addressing y-absolute y-window	Values as above, but with substitution of "y" for the "x" dimension references except that when y-bound is "unbounded" the default value for y-window = 1. Valid if dimensions = 2 or 3.
z-bound z-addressing z-absolute z-window	Values as above, but with substitution of "z" for the "x" dimension references except that when z-bound is "unbounded" the default value for z-window = 1. Valid if dimensions = 3.

If blocks or fields are used for a DO, the X-dimension must be bounded; the value "unbounded" for VTE-parameter *x-bound* is not valid and no default value is defined in this case.

The VTE-parameter **x-addressing** defines whether the value of the display object pointer for that dimension can be altered by an explicit addressing operation. Change to a lower value is not possible if the value is "higher only". If the value is "not permitted" no explicit change is allowed.

The VTE-parameter **x-absolute** defines whether the value of the display object pointer for that dimension can be set to a new value by a pointer-absolute addressing operation; such change is also subject to *x-addressing*. Value "yes" means that the coordinate can be set to any valid value.

The VTE-parameter **x-window** defines the update-window-size for the x-dimension (see 19.1.1.3).

If block-definition-capability (13.1.2) or field-definition-capability (13.1.3) is selected for a display object, the VTE-parameter *x-window* is not used for such a DO (it assumes the default value of *x-bound* value appropriate for a bounded dimension).

The above definitions apply to the Y dimension VTE-parameters with the substitution of y for x. The above definitions apply to the Z dimension VTE-parameters with the substitution of z for x except that there are no restrictions on *z-bound* and *z-window* due to block-definition-capability or field-definition-capability selection.

18.2.4 Repertoire-assignment VTE-parameters

The VTE-parameters in table 4 enable a repertoire or repertoires to be assigned and a set of fonts to be associated with each such repertoire.

The two VTE-parameters specified in table 4 may be defined once for each repertoire indicated as available by the integer value of the VTE-parameter *repertoire-capability*. Multiple occurrences form an ordered list where individual occurrences (i.e., repertoires) are referenced by their positions in the ordered list.

Table 4 - Repertoire VTE-parameters

Parameter	Value
repertoire-assignment	Each instance designates one discrete repertoire; the instances form an ordered list as defined in 18.2. (Default = <ESC> 2/8 4/0; the default repertoire consists only of the set of graphic characters of the IRV of ISO/IEC 646 used as in a) in 18.2.4.1).
font-capability	Each instance of this VTE-parameter is an integer 1..N specifying the number of fonts available for the repertoire with which this instance is associated (default = 1 for each implied instance).
NOTE – The inclusion of <ESC> in the references to escape sequences is for clarity only; the actual <ESC> character is omitted from values of repertoire-assignment.	

A value for an instance of *repertoire-assignment* VTE-parameter consists of two components; these are not treated as separate VTE-parameters and are not shown in figure 7. These components are repertoire-assignment-type and repertoire-assignment-value.

Repertoire-assignment-type defines the method used to designate the repertoire and determines the form of **repertoire-assignment-value**. It is optional and if present is of type ASN.1 OBJECT IDENTIFIER.

This International Standard defines a single standard *repertoire-assignment-type* of value {vt-b-rep-iso2022}, see annex C, which is the default type if this component is omitted. *Repertoire-assignment-value* for this type is defined in 18.2.4.1.

18.2.4.1 Repertoire-assignment-value for {vt-b-rep-iso2022}

When *repertoire-assignment-type* is absent or has this value the repertoire designation uses the principles of ISO/IEC 2022.

A single discrete repertoire (see clause B.12) is thus composed of the following components as defined in ISO/IEC 2022:

a) a G set usable in the GL position of the code table; mandatory;

b) a G set usable in the GR position of the code table; optional;

c) a C0 set; optional;

d) a C1 set; optional (see clauses B.12 and B.13).

NOTE 1 – GL position refers to the left graphics partition defined in ISO/IEC 2022, i.e., columns 02...07 (excluding 2/0 and 7/15) of the 8-bit code space and GR similarly refers to the right graphics partition, i.e., columns 10...15 (excluding 15/15). The code table positions excluded have a fixed meaning in ISO/IEC 2022.

For such a repertoire, an instance of *repertoire-assignment-value* is a sequence of up to four items, each of which is an escape sequence as defined in ISO/IEC 2022 to designate a coded character set (omitting the <ESC> character). These correspond (not necessarily in order) to items in the above list. For the GL position the G0 designation sequence must be used. For the GR position, the G1 designation sequence must be used.

Alternatively a repertoire may consist only of the Virtual Terminal Service Transparent Set. This is a 256 element set specially registered for this International Standard. It consists of 256 values 0...255 with no assigned meanings in the VT Service. It has the ISO Registration Number of 125. In this case, *repertoire-assignment-value* is the single (escape) sequence 2/5 2/15 4/2 as defined for this register entry.

If *repertoire-assignment* is omitted, this repertoire-assignment-type applies by default and the default repertoire of this type is that corresponding to the single (escape) sequence 2/8 4/0, i.e., the set of graphic characters (only) of the IRV of ISO/IEC 646.

Inclusion in a repertoire of a G-set which includes non-spacing diacritical marks implies that the valid combinations of these with other graphic characters, as defined in the registered code set or in a document referenced by the register entry, are available in the repertoire as storable characters occupying single array elements. Support of such storage is implied by acceptance of such a repertoire. No other combinations using the non-spacing marks are available, nor can they be used alone – use with space may be available in the character set definition.

If a repertoire includes a G-set which is defined in the register as a multi-byte set, this implies that the repertoire includes all the valid combinations as defined in the register, as separate storable characters occupying single array elements. Support of such storage is implied by acceptance of such a repertoire. No other combinations are permitted.

The VT Service does not support, as part of a repertoire, compound characters formed using the <BS> (i.e., backspace) control of ISO/IEC 646 to achieve superimposition of two graphic characters (see B.12 and B.13.)

If the mandatory G set is a 94 character set, it is implicit that the SPACE character is available in position 2/0 of the GL code table.

NOTES

2 For the meanings of the character set related terms used in this sub-clause see ISO/IEC 2022, which also gives the general rules for use of registered code sets. For the definition of the character sets designated by particular escape sequences, refer to the International Register of Coded Character Sets to be Used with Escape Sequences, which also gives any particular rules for the use of a set.

3 Registration Number refers to the numbers assigned by the registration authority (ECMA) in the International Register of Coded Character Sets to be Used with Escape Sequences.

4 The list of fonts available for each repertoire is specified by corresponding occurrences of the VTE-parameter *font-assignment*, see 18.3

18.2.5 Colour Assignment VTE-parameters

The VTE-parameters in table 5 enable two sets of colour values to be assigned for use separately as foreground and background colour rendition secondary attributes.

The maximum number of instances for these VTE-parameters is the value of the corresponding "capability" VTE-parameter (see 18.1); for each of *foreground-colour-assignment* and *background-colour-assignment*, the instances form an ordered list and each occurrence may be referenced by its position in the list.

A default value is defined only for the case where a "capability" VTE-parameter has value 1; the default for the assignment VTE-parameter is then "device-dependent".

NOTE – The value "device-dependent" cannot be explicitly negotiated and has no representation.

A value for an instance of either of the colour-assignment VTE-parameters in table 5 consists of two components; these are not treated as separate VTE-parameters and are not

Table 5 - VTE-parameters for Colour Assignment

Parameter	Value
foreground-colour-assignment	each instance designates one colour assignment; the instances form an ordered list as defined in 18.2 (default = "device-dependent" for the case where *foreground-colour-capability* has the value 1).
background-colour-assignment	as for *foreground-colour-assignment* (default = "device-dependent" for the case where *background-colour-capability* has the value 1).

shown in figure 7. These components are colour-assignment-type and colour-assignment-value.

Colour-assignment-type defines the method used to designate the colour value and determines the form of **colour-assignment-value**. It is optional and if present is of type ASN.1 OBJECT IDENTIFIER.

A value of *colour-assignment-type* applies to any following instances of *colour-assignment-value* in the same ordered list which do not have an explicit *colour-assignment-type* component (this avoids the need for duplication of this component in lists of the same type). This modal application is terminated at the end of an ordered list of instances of this VTE-parameter.

This International Standard defines a single standard *colour-assignment-type* of value {vt-b-colour-iso6429}, see annex C, which is the default type if this component is omitted. *Colour-assignment-value* for this type is defined in 18.2.5.1.

18.2.5.1 Colour-assignment-value for {vt-b-colour-iso6429}

When *colour-assignment-type* is absent or has this value, the colour value designation uses the (English) colour names given to the appropriate parameter values for SET GRAPHIC RENDITION in ISO/IEC 6429.

18.2.6 VTE-parameter for Emphasis

A single VTE-parameter is provided as shown in table 6.

It is optional and can occur as multiple occurrences which form an ordered list; each occurrence may be referenced by its position in this list.

This International Standard does not define a significance for the value of an occurrence of this VTE-parameter, which is provided so that a VTE-profile definition can provide a VTE-profile argument to allow negotiation of a specific display object emphasis functionality from a more generic functionality in the VTE-profile definition. See also clause B.17.

NOTE – Such provision in a VTE-profile definition is optional.

Table 6 - Emphasis VTE-parameter

Parameter	Value
DO-emphasis	each instance is of type ASN.1 PrintableString

18.3 Tertiary VTE-parameter

The optional **font-assignment** VTE-parameter is specified in table 7. Multiple instances of this tertiary VTE-parameter may be defined. Its occurrences are controlled by the *repertoire-capability* and *font-capability* VTE-parameters.

Table 7 - Font-assignment VTE-parameter

Parameter	Value
font-assignment	each instance designates one font assignment; the instances form an ordered list as defined in 18.2 (default = "device-dependent" for the case where *font-capability* has the value 1)

The value of *repertoire-capability* implies a list of repertoires as defined in 18.1 and 18.2.4. With each of these repertoires is associated an instance of *font-capability* as defined in table 4. Each such instance implies an ordered list of *font-assignment* VTE-parameter values, the fonts being identified by their position in the list. Not all the implied entries need to be given an explicit assignment. The lists themselves are in an ordered list of the same ordering as the repertoire assignments. Both individual list entries and complete lists may be replaced by a "null" value to act as a placeholder when it is not required to give an explicit value.

A value for an instance of *font-assignment* VTE-parameter consists of two components; these are not treated as separate VTE-parameters and are not shown in figure 7. These components are font-assignment-type and font-assignment-value.

Font-assignment-type defines the method used to designate the font and determines the form of **font-assignment-value**. It is optional and, if present, is of type ASN.1 OBJECT IDENTIFIER.

A value of *font-assignment-type* applies to any following instances of *font-assignment-value* which do not have an explicit *font-assignment-type* component (this avoids the need for duplication of this component in lists of the same type). This modal application is terminated at the end of an ordered list of instances of this VTE-parameter.

This International Standard does not define a standard font designation method. It defines a single value for *font-assignment-type* of value {vt-b-font-adhoc}, see C.2, which enables the VT-users to supply a font name in the form of an ASN.1 PrintableString. This type is the default if an explicit type is not present.

NOTE – For the default value "device-dependent", the device may use whatever font it has available, see B.14.

19 Operations on display objects

19.1 Addressing operations

19.1.1 Addressing operations with the primitive display pointer

19.1.1.1 Implicit addressing with the primitive display pointer

When a value is written to the primary attribute of an array element and ripple is not enabled, the x-coordinate value of the primitive display pointer is automatically incremented by one. This mechanism is automatically disabled when the x-coordinate value reaches a value one greater than the x-dimension bound. Constraints on use of update operations when the display pointer takes such a value are defined in 19.4. This mechanism may be re-enabled by explicit adjustment of the display pointer x-coordinate to a lower value. The mechanism is never disabled if the x-dimension is unbounded.

When a value is written to a primary attribute value and ripple is enabled, the display pointer is updated to point to the next element of the ripple-extent (or to point immediately after the ripple-extent, if there is no next element). See 19.2

19.1.1.2 Explicit addressing with the primitive display pointer

Operations are provided for the explicit modification of the value of the primitive display pointer.

An explicit address operation is invalid if any of its implicit or explicit arguments violates the constraints applied by the VTE-parameters *d-bound*, *d-addressing* and *d-absolute* for any applicable dimension d (d = x, y or z), see 18.2.3.

The use of these operations does not affect the value of the content of any previously existing array element nor does it generate any new array elements. Updating an array element does not automatically initiate any of these explicit addressing operations.

19.1.1.2.1 primitive operations: these operations permit the display pointer value to be changed to any value valid for all defined dimensions. They are independent of update-window bounds.

a) POINTER-ABSOLUTE target-address

where target-address is either

- an explicit display pointer value (x_t,y_t,z_t) where each coordinate is optional and only valid if the corresponding dimension is defined; if a coordinate is omitted, the current value is retained for that dimension, or
- one of the following special values defined in 19.1.1.4: "start", "start-y", "start-x" "end", "end-y", "end-x".

The operation sets the display pointer to the value of target-address.

b) POINTER-RELATIVE p, q, r

where p, q and r are signed integers specifying the required change to x, y and z coordinates of the display pointer; each is optional (default = 0, no change) and only valid if the corresponding dimension is defined. The operation sets the display pointer as follows:

$$x := x + p; \quad y := y + q; \quad z := z + r.$$

19.1.1.2.2 macro operations: the following operations have no arguments and may only be used in the form stated:

a) NEXT X-ARRAY $(y := y + 1; x := X_{min})$;
b) PREVIOUS X-ARRAY $(y := y - 1; x := X_{min})$;
c) NEXT Y-ARRAY $(z := z + 1; y := Y_{min}; x := X_{min})$;
d) PREVIOUS Y-ARRAY $(z := z - 1; y := Y_{min}; x := X_{min})$;

where X_{min} , Y_{min} are the lower update-window bounds for the destination X-array and Y-array respectively.

NOTE – The left-to-right order for the macro operations is necessary to ensure correct operation as X_{min} is a function of y and z, and Y_{min} is a function of z (see 19.1.1.3).

19.1.1.3 Update-window

Each defined dimension of a display object has an associated VTE-parameter *d-window*, see 18.2.3, which enables the VT-users to agree update constraints in addition to those implied by dimension bounds or display pointer movement capabilities. The arrays of a dimension at differing coordinate values of the next higher dimension each have an independent update-window which can move during update operations on the display object. Clause B.3 discusses the use of update-window by implementations.

The update-window places lower limits on locations which may be updated by TEXT, REPEAT-TEXT, ATTRIBUTE or ERASE operations on a display object. It does not, however, in itself constrain the upper limit of TEXT operations, which can cause the update-window to move to encompass a set of higher coordinate values. See 19.4.1.2 for effect on and of REPEAT-TEXT operation.

An explicit value for update-window-size can be negotiated independently for each defined array dimension (see clause B.3).

For a bounded dimension, the update-window-size may have values defined by *d-window* and the array bounds such that

$$0 \leq \text{update-window-size} \leq \text{array-bound for the dimension,}$$

with a default value equal to the array-bound; for an unbounded dimension there is no upper limit on update-window-size with a default value of 0.

For any dimension, let W be the value of *d-window* and let H be the highest d-coordinate value at which a primary attribute value has ever been set in the selected d-array. Then the upper bound of the update-window is

$$U = \max (H, W)$$

and the lower bound is

$$L = U - W + 1$$

The update-window does not constrain the value of the display pointer.

In the special case of a bounded dimension for which update-window-size, explicitly or by default value, equals the dimension bound, the update window is always positioned with the lower bound at coordinate 1 and the upper bound at the dimension bound so that all dimension values are within the update-window. Thus the term "within the update-window bounds" used in defining various operations implies all coordinate values from 1 to the bound inclusive.

In the general case, there is an independent lower update-window bound for each X-array; the lower update-window bound is a function of y and z coordinate values.

Similarly the lower update-window bound for a Y-array is a function of the z coordinate value.

NOTES

1 This mechanism requires the service provider to keep a record of the coordinate of the highest updated array element in a dimension for each valid value of the next higher order dimension at which any updates have been made.

2 The update-window for an array can move forward only in response to TEXT or REPEAT-TEXT operations updating array elements higher than those previously updated, but not as a result of merely setting the display pointer to a higher value.

3 If the update-window does move forward, so that the lower update-window bound becomes greater than 1, array elements below the lower bound become permanently inaccessible for update.

4 An update-window size of zero prohibits backward updates to elements of arrays on the dimension.

19.1.1.4 Address extents for multiple array element operations

Operations are defined in 19.4 which can affect a number of array elements, this being determined by the *address extent* for the operation.

An **address extent** is specified by a start address (x_s, y_s, z_s) and a finish address (x_f, y_f, z_f), where

$(x_s, y_s, z_s) \leq (x_f, y_f, z_f)$, see 13.1.1.

The address extent includes all addresses (x_i, y_i, z_i) for which

$(x_s, y_s, z_s) \leq (x_i, y_i, z_i) \leq (x_f, y_f, z_f)$.

If a particular dimension is not defined, its coordinate value is ignored in the above condition statements (or can be considered to have the fixed value 1 (one)).

For the ATTRIBUTE and ERASE operations (19.4.1.3 and 19.4.1.4), an address extent is deemed to include only those coordinate combinations which are within the update-windows of all dimensions although an extent specification can include values outside this range.

The following special display pointer values are given symbolic names :

"current"	: (x_c, y_c, z_c)	the current value of the display pointer;
"start"	: (1,1,1)	)
"start-y"	: $(1, 1, z_c)$	) can only be used as extent start;
"start-x"	: $(1, y_c, z_c)$	)
"end"	: (x_m, y_m, z_m)	)
"end-y"	: (x_m, y_m, z_c)	) can only be used as extent end;
"end-x"	: (x_m, y_c, z_c)	)

where x_c, y_c, z_c represent the coordinate values of the current display pointer and x_m, y_m, z_m, are the upper bounds of the update-window of the dimensions, see 19.1.1.3.

These can be used in any pair or with an explicit (x,y,z) value to define an address extent.

Use of the "current" form as start or finish is invalid if x_c is greater than x-bound (see 19.1.1.1).

The special display pointer symbolic values are valid if one or more dimensions are undefined, with the appropriate coordinate being omitted.

NOTES

1 x_m is a function of the y and z coordinates, and y_m is a function of the z coordinate.

2 Some address operations in 19.1.1.2 refer to some of the above special display pointer values as argument values.

19.1.2 Addressing operations with extended display pointer

19.1.2.1 Implicit addressing with extended display pointer

When blocks are in use for a DO and ripple is not enabled, implicit addressing operates for the p coordinate of the extended display pointer (see 13.1.1.1) as defined in 19.1.1.1 for the x-coordinate of the primitive display pointer, except that the p-coordinate automatic increment is disabled when the p-coordinate reaches a value one greater than the width Dx of the current block. The implicit mechanism can be re-enabled by explicit adjustment of the p-coordinate to a lower value, or by a change of b or z-coordinate to address a block in which this value of p is valid. The reference in 19.1.1.1 to an unbounded x-dimension is not applicable when block-definition-capability is used.

When a value is written to a primary attribute value and ripple is enabled, the display pointer is updated to point to the next element of the ripple-extent (or to point immediately after the ripple-extent, if there is no next element). See 19.2.

19.1.2.2 Explicit addressing with extended display pointer

When blocks are in use for a DO, the explicit addressing operations of 19.1.1.2 are amended as follows:

a) an explicit addressing operation is valid even if the resulting p and q coordinate values are not within the bounds of the block addressed by the resulting b and z coordinate values, or if there is no defined block (however, no update to an array element can be performed);

b) any explicit addressing operations may be used to alter the value of b coordinate since no constraints are applied to movement on the b-dimension.

19.1.2.2.1 Primitive operations

When blocks are in use for a DO, the primitive operations defined in 19.1.1.2.1 are amended to the following:

a) POINTER-ABSOLUTE target-address

where target-address is either

- an explicit pointer value (p_t,q_t,b_t,z_t) where each coordinate is optional, and q_t and z_t are only valid if the corresponding (Y respectively Z) dimension is defined; if a coordinate is omitted the current value is retained for that dimension, or
- one of the following special values defined in 19.1.2.5:
  "start", "start-b", "start-q", "start-p",
  "end", "end-b", "end-q", "end-p".

The operation sets the extended display pointer to the value of target-address.

b) POINTER-RELATIVE relative-target-address

where relative-target-address has components m, n, s and r which are signed integers specifying the required change to p, q, b and z coordinates respectively of the extended display pointer; each is optional, default = 0 (no change), and n and r are valid only if the Y and Z dimensions respectively are defined.

The operation sets the display pointer as follows:

$$p_t := p_c + m;\quad q_t := q_c + n;\quad b_t := b_c + s;\quad z_t := z_c + r$$

where $_c$ refers to the initial value of the display pointer and $_t$ to the resulting value.

19.1.2.2.2 Macro operations

When blocks are in use for a DO, the following "macro" operations are available and can be used only in the form stated; in addition to operations derived from those in 19.1.1.2.2, there are two additional operations.

Derived operations:

a) NEXT X-(SUB)ARRAY (P-array) (q:=q+1; p:=1);
b) PREVIOUS X-(SUB)ARRAY (P-array) (q:=q-1; p:=1);
c) NEXT Y-ARRAY (z:=z+1; b:=1; q:=1; p:=1);
d) PREVIOUS Y-ARRAY (z:=z-1; b:=1; q:=1; p:=1).

Additional operations:

e) NEXT BLOCK (b:=b+1; q:=1; p:=1);
f) PREVIOUS BLOCK (b:=b-1; q:=1; p:=1).

NOTE – The definitions of the derived operations do not refer explicitly to P and Q equivalents of Xmin or Ymin as in 19.1.1.2.2 since by the restriction in 13.1.2 these values cannot be other than 1.

19.1.2.3 Update-window mechanism

When blocks are in use for a DO only the Z-window is effective, see 18.2.3.

19.1.2.4 Address values and update constraints

The extended display pointer can be set to a coordinate combination for which z and b do not address an existing block or for which q or p is invalid for the addressed block. This assignment is valid. No TEXT, ATTRIBUTE or ERASE update operations can be performed.

19.1.2.5 Address extents with extended display pointer

When blocks are in use for a DO, the address extents defined in 19.1.1.4 are amended and extended to operate as follows.

An address extent is specified by a start address (p_s, q_s, b_s, z_s) and a finish address (p_f, q_f, b_f, z_f), where

$$(p_s, q_s, b_s, z_s) \leq (p_f, q_f, b_f, z_f),$$ see 13.1.2.

The address extent includes all addresses (p_i, q_i, b_i, z_i) for which

$$(p_s, q_s, b_s, z_s) \leq (p_i, q_i, b_i, z_i) \leq (p_f, q_f, b_f, z_f).$$

If a particular dimension is not defined, its coordinate value is ignored in the above condition statements (or can be considered to have the fixed value 1 (one)).

For the ATTRIBUTE and ERASE operations (19.4.1.3 and 19.4.1.4), an address extent is deemed to include only those coordinate combinations which are within the update-windows

of all dimensions although an extent specification can include values outside this range.

An address extent can include (b,z) coordinate combinations at which no block currently exists. Such values are skipped.

NOTE 1 – Due to overlap of blocks, an array element may be updated more than once by one address extent.

The following special display pointer values are given symbolic names:

"current"	: (p_c, q_c, b_c, z_c)	the current value of the extended display pointer;
"start"	: (1,1,1,1)	)
"start-b"	: (1,1,1,z_c)	) can only be used
"start-q"	: (1,1,b_c,z_c)	) as extent start;
"start-p"	: (1,q_c,b_c,z_c)	)
"end"	: (p_m,q_m,b_m,z_m)	)
"end-b"	: (p_m,q_m,b_m,z_c)	) can only be used
"end-q"	: (p_m,q_m,b_c,z_c)	) as extent end;
"end-p"	: (p_m,q_c,b_c,z_c))	)

where:

- p_c, q_c, b_c, z_c represent the current coordinate values of the extended display pointer;
- p_m, q_m represent the maximum p and q coordinates in a block and are a function of a (b, z) coordinate combination;
- b_m represents the maximum value of b at which a block exists on a Y-array and is a function of z coordinate;
- z_m is the upper bound of the update-window on Z dimension.

These can be used in any pair or with an explicit (p,q,b,z) value.

Use of the "current" form as start or finish is invalid if p coordinate is above the maximum value for the current block, see 19.1.2.1.

The special display pointer symbolic values are valid if one or more dimensions are undefined, with the appropriate coordinate being omitted.

NOTE 2 – Some address operations in 19.1.2.2 refer to some of the above special display pointer values as argument values.

19.1.3 Logical addressing operations with logical pointer

Logical addressing operations are used to update the logical pointer and are applicable if fields are in use for a DO.

19.1.3.1 Implicit logical addressing

An implicit addressing mechanism is provided for the K-dimension within a field analogous to that for the X-dimension, see 19.1.1.1. Whenever a value is written to the primary attribute of an array element contained within a field (using LOGICAL-TEXT operation, 19.4.2.1) and ripple is not enabled, the k-coordinate of the logical pointer is incremented by one. This mechanism is automatically disabled when k reaches a value one greater than the upper bound of the field addressed by coordinates (f,z) of the logical pointer. The implicit mechanism can be re-enabled by explicit adjustment of the k-coordinate to a lower value, or by a change of f or z coordinate to address a field in which this value of k is valid.

When a value is written to a primary attribute value and ripple is enabled, the display pointer is updated to point to the next element of the ripple-extent (or to point immediately after the ripple-extent, if there is no next element). See 19.3.

19.1.3.2 Explicit logical addressing

Primitive and macro operations are provided for the explicit modification of the logical pointer.

The use of these operations does not affect the value of the content of any array element. No entry of a value into any array element can automatically initiate any of these explicit addressing operations.

19.1.3.2.1 Primitive logical addressing operations

The following primitive operations are defined:

a) LOGICAL-ABSOLUTE target-logical-address

where target-logical-address is either

- an explicit logical pointer value (k_t,f_t,z_t) where each coordinate is optional and z_t is valid only if Z dimension is defined; if a coordinate is omitted the current value of the dimension is retained, or
- one of the following special values defined in 19.1.3.5:
  "log-start", "start-f", "start-k",
  "log-end", "end-f", "end-k".

The operation sets the logical pointer to the value of target-logical-address.

b) LOGICAL-RELATIVE relative-target-logical-address

where relative-target-logical-address has components u, v, and w which are signed integers specifying the required change to k, f and z coordinates respectively of the logical pointer; each is optional, default = 0 (no change), and w is valid only if the Z dimension is defined.

The operation sets the logical pointer as follows:

$k_t := k_c + u$; $f_t := f_c + v$; $z_t = z_c + w$

where $_c$ refers to the initial value of the logical pointer and $_t$ refers to the resulting value.

19.1.3.2.2 Macro logical addressing operations

The following "macro" operations are available and can be used only in the form stated:

a) NEXT-FIELD (k: = 1; f: = next(f));
b) PREVIOUS-FIELD (k: = 1; f: = previous(f));

where next(f) and previous(f) are derived with reference to the next-field and previous-field entries in the FDR for the field as given in 20.3.3.4.

19.1.3.3 Update-window mechanism

When fields are in use for a DO only the Z-window is effective, see 18.2.3.

19.1.3.4 Address values and update constraints

The logical pointer can be set to a combination of z and f coordinates referring to a non-extant or an inactive field, see 20.3.3.5. The logical pointer can also be set to have a k value which is invalid for the field selected by the z and f values. Such assignments are valid. No logical update operation, including ATTRIBUTE and ERASE operations, can be performed to update a location referenced by such values.

19.1.3.5 Address extents with field-definition-capability

A logical address extent is used with a number of operations on fields defined later in this clause.

A **logical address extent** is specified by a start logical address (k_s, f_s, z_s) and a finish logical address (k_f, f_f, z_f), where

$(k_s, f_s, z_s) \leq (k_f, f_f, z_f)$, see 13.1.3.1.

The address extent includes all addresses (k_i, f_i, z_i) for which

$(k_s, f_s, z_s) \leq (k_i, f_i, z_i) \leq (k_f, f_f, z_f)$.

For the LOGICAL ATTRIBUTE and LOGICAL ERASE operations (19.4.2.3 and 19.4.2.4), an address extent is deemed to include only those coordinate combinations which are within the update-windows of all dimensions although an extent specification can include values outside this range.

A logical address extent may include locations contained in non-extant or inactive fields, see 20.3.3.5. Logical operations using such an address extent shall skip these locations.

The following special values are given symbolic names:

"log-current"	: (k_c, f_c, z_c)	the current value of the logical pointer;
"log-start"	: (1,1,1)	)
"start-f"	: (1,1, z_c)	) can only be used as extent start;
"start-k"	: (1, f_c, z_c)	)
"log-end"	: (k_m,f_m,z_m)	)
"end-f"	: (k_m,f_m,z_c)	) can only be used as extent end;
"end-k"	: (k_m,f_c,z_c)	)

where:

- k_c, f_c, z_c represent the current coordinate values of the logical pointer;
- k_m represents the maximum k coordinate in a field and is a function of a (f, z) coordinate combination;
- f_m represents the maximum value of f at which a field exists on a Y-array and is a function of z coordinate;
- z_m is the upper bound of the update-window on Z dimension.

These can be used in any pair or with an explicit (k,f,z) value. Use of the "log-current" form as start or finish is invalid if k coordinate is above the maximum value for the current field, see 19.1.3.1.

The special logical pointer symbolic values are valid if Z dimension is undefined, with this coordinate being omitted.

NOTE – Some of the explicit logical addressing operations defined in 19.1.3.2.1 refer to some of the above special logical pointer values as argument values.

19.2 Ripple operations

Ripple operations provide for the bulk movement of character box elements either to create space for an insertion or to close up a space after a deletion. Ripple operations are not provided as separate DO updates. Instead, they are used as part of the definition of DO updates such as TEXT, COPY-FROM-BUFFER and INSERT-X-ARRAY when ripple is enabled.

Ripple is enabled when the DO has *ripple-capability* "yes" (see 18.1) and the ripple-mode element of the RMCO is "true" (see 14.6 and 20.3.9).

A ripple operation is specified by defining:

- the elements affected by the ripple (the ripple-extent, see 19.2.1);
- the ripple coordinate (x, y or z);
- ripple direction (forwards for insertion, backwards for deletion);
- the number of units of ripple, N.

The coordinate, direction and number of units depend on the DO update performing the ripple operation.

In a ripple operation, the ripple-extent is taken as a sequence of units, the unit depending on the coordinate direction of the ripple:

x	array element;
y	x-array;
z	y-array.

The content of a unit is the collection of primary and secondary attributes of all the array elements of the unit.

For a forward ripple, the content of each unit M after the Nth is set to the previous content of unit M – N. The new content of the first N units is specified by the DO update initiating the ripple. If the ripple extent is bounded, the old content of the last N units is lost.

For a backward ripple, the content of each unit M except the last N is set to the previous content of unit M + N. If the ripple-extent is bounded, each array element of the last N units is set according to the fill procedure, (see 19.2.2).

19.2.1 Definition of ripple-extent

The ripple-extent for an operation consists of the extent for that operation, extended either to an end point determined by ripple-limit or to any array element which is below any update-window, whichever is smaller. The end point determined by

the value of the ripple-limit element in the RMCO is the end of an array containing the end of the operation extent:

	ripple-limit	array
primitive operations		
	x	x-array
	y	y-array
	z	z-array
extended operations		
	p	p-array
	q	block
	b	extended-y-array
	z	extended-z-array

NOTE – The above definition ensures that the ripple-extent does not contain any array elements which are not updatable because of update-window constraints, unless they are already contained in the operation extent (in which case the operation is invalid).

19.2.2 Fill operations

Filling determines the primary and secondary attributes of array elements which are emptied by ripple operations.

If the fill-mode element of the RMCO (20.3.9) has value "erase", the primary attribute is unset and all secondary attributes are set to the explicit modal default.

If the fill-mode element of the RMCO has value "fill", the primary attribute is set to the value of the fill-character element of the RMCO. The secondary attribute character-repertoire is set from the *CO-repertoire* VTE-parameter of the fill-character element of the RMCO. Each rendition secondary attribute is determined by the following conditions, in order:

a)

1) if an operation using the display pointer is being performed, the corresponding modal attribute if not "null";
2) if an operation using the logical pointer is being performed, the corresponding FDR attribute if set, else the field modal attribute if not "null";

b) the corresponding global attribute value if not "null";

c) the explicit modal default.

19.3 Logical ripple operations

Logical ripple operations provide for the bulk movement of character box elements either to create space for an insertion or to close up space after a deletion. Logical ripple operations are not provided as separate logical DO updates. Instead, they are used as part of the definition of logical DO updates such as LOGICAL-TEXT and COPY-LOGICAL-FROM-BUFFER when ripple is enabled.

Logical ripple is enabled when the DO has *ripple-capability* "yes" and the ripple-mode element of the RMCO is "true".

A logical ripple operation is specified by defining:

- the elements affected by the logical ripple (the ripple-extent) see 19.3.1;
- the ripple direction (forwards for insertion, backwards for deletion);
- the number of units of ripple, N.

The ripple direction and the number of units of ripple depend on the logical DO update performing the ripple operation.

In a logical ripple operation, the ripple-extent is taken as a series of units, the content of each being the collection of primary and secondary attributes of the single array element, i.e., the ripple coordinate takes the value "k".

For a forward ripple, the content of each unit M after the Nth is set to the previous content of unit M – N. The new content of the first N units is specified by the logical DO update initiating the ripple. If the ripple-extent is bounded, the old content of the last N units is lost.

For a backward ripple, the content of each unit M except the last N is set to the previous content of unit M+N. If the ripple-extent is bounded, each array element of the last N units is set according to the fill procedure, see 19.2.2.

19.3.1 Ripple-extent for logical operations

The ripple-extent for a logical operation consists of the logical extent for that operation extended to an end point determined by the value of logical-ripple-limit element of the RMCO.

If logical-ripple-limit = "k", the end point is the end of the field containing the end of the operation extent.

If logical-ripple-limit = "f", the end point is the last active field on the f-dimension of the y-array containing the end of the operation extent.

If the logical-ripple-limit = "z", the end point is the last active field defined (i.e., the field with the highest (f,z) coordinate).

19.4 Update operations

19.4.1 Update operations using display pointer

This sub-clause defines the update operations which are available for updating the content of array elements of a display object using the primitive display pointer or the extended display pointer, see 13.1.1.1. All these operations are subject to the access-rule for the display object. They may also be restricted by *access-outside-fields*, see 18.2.2 and 19.5.

19.4.1.1 TEXT operation

This operation has the form

TEXT primary-attribute-value

It enters a specified primary attribute value into the array element currently identified by the display pointer and invokes implicit addressing (see 19.1.1.1 and 19.1.2.1).

This operation is subject to access control as defined in 19.5.

The single argument **primary-attribute-value** should be in the range appropriate to the secondary attribute *character-repertoire* for the array element, which is determined by the following conditions examined in the order given:

a) modal attribute value for this attribute if it is not "null";

b) the value of this attribute in the array element if the element is not empty;

c) the explicit modal default for this attribute (see 18.2).

The secondary attribute *character-repertoire* value in the array element is set to the value chosen (i.e., no change in case b)). (See also *repertoire-capability* and *repertoire-assignment* VTE-parameters in 18.1 and 18.2.4).

NOTE – This International Standard does not define the significance of a primary attribute value which has no assigned meaning in the applicable repertoire.

The value for each rendition secondary attribute (see 13.2) in the array element after the TEXT operation is determined by the following conditions, taken in order:

d) the corresponding modal attribute value if it is not "null";

e) an existing value in the array element is left unchanged if it is not "null";

f) the corresponding global attribute value if it is not "null";

g) the corresponding explicit modal default (see 18.2);

except that for secondary attribute *font*, if the TEXT operation changes the secondary attribute character-repertoire, then conditions e) and f) do not apply and the value for font in g) cannot be determined until character-repertoire is known.

The TEXT operation is invalid if the display pointer does not currently identify an array element, see 19.1.1.1, or if the identified array element is below a lower update-window bound, see 19.1.1.3.

If ripple is enabled (19.2), a forward ripple is performed before the primary and secondary attributes are updated, with ripple coordinate x and ripple of one unit. The ripple-extent is derived as specified in 19.2.1 from an operation extent consisting of the single array element at the display pointer.

19.4.1.2 REPEAT-TEXT operation

This operation has the form

REPEAT-TEXT finish-address
primary-attribute-value-string

It has the effect of repeating a TEXT operation (19.4.1.1) for array elements within the repeat-extent as defined below.

The arguments are

a) **finish-address** is combined with "current" to produce repeat-extent; this is a subset of address-extent as defined in 19.1.1.4 or 19.1.2.5. The start address is always "current" and the finish address must be greater than or equal to "current" (if these are equal the operation is exactly equivalent to TEXT operation).

b) **primary-attribute-value-string** is a sequence of one or more primary attribute values as defined for the TEXT operation in 19.4.1.1.

This operation is subject to access control as defined in 19.5.

The REPEAT-TEXT operation operates on each character-box graphic element which is within the repeat-extent and which, for each dimension d, is either within the update-window for its d-array or within the last d-array of the repeat-extent and is above the lower bound of the update-window.

For each array element updated, a primary attribute value is taken from primary-attribute-value-string, starting with the first value in the sequence, continuing through the sequence and at the end restarting at the start of the sequence. If any value is invalid for the secondary attribute character-repertoire, determined as defined in 19.4.1.1 a)...c), the primary attribute value in the updated array element is undefined. Other secondary attributes of updated array elements are determined as defined in 19.4.1.1 d)...g).

After a REPEAT-TEXT operation, the display pointer identifies the position immediately after the last array element updated.

If ripple is enabled (19.2), a forward ripple is performed before the REPEAT-TEXT operation, with ripple coordinate x and the number of units equal to the number of array elements in the repeat-extent. The ripple-extent is derived from the repeat-extent as specified in 19.2.1.

NOTE – If blocks are in use for the DO, blocks may overlap and this can result in an array element being updated more than once.

19.4.1.3 ATTRIBUTE operation

This operation has the form

ATTRIBUTE attribute-id attribute-value attribute-extent

It sets the secondary attribute selected by attribute-id to the value designated by attribute-value for some or all array elements, or sets the corresponding modal or global attribute value, or certain combination actions, according to the value of attribute-extent.

The arguments are:

a) **attribute-id** is the identifier for one of the secondary attributes listed in 13.2;

b) **attribute-value** is a permitted value for this attribute, see d)..f), or can be "null" if attribute-extent is "modal" or "global", see c);

c) **attribute-extent** takes one of the following forms:

1) any valid address-extent as defined in 19.1.1.4 or 19.1.2.5;

2) "global": implicitly selects address extent "start","end" for direct action, and the global attribute value is set to attribute-value; attribute-value "null" is valid;

3) "modal": the modal attribute value is set to attribute-value (no array elements are selected for direct action by this value of attribute extent); attribute-value "null" is valid.

This operation is subject to access control as defined in 19.5.

Permitted attribute values are:

d) for character-repertoire, foreground-colour or background-colour: an integer from 1 up to the value of the corresponding *xxx-capability* VTE-parameter, see 18.1;

e) for font: an integer greater than or equal to 1;

f) for emphasis: an ASN.1 PrintableString; see B.17.

If attribute-id has value "character-repertoire" the only valid value for attribute-extent is "modal".

If the modal value for character-repertoire attribute is changed, the modal value for font is set to the explicit modal default value for font for the new value of character-repertoire (to avoid incompatible values for these two attributes).

If attribute-id has value "font" and attribute-extent has value "modal", then attribute-value must be a value which is valid for the current modal value for character-repertoire.

If the value of font being applied by an extent other than "modal" is, for any array element, not valid for the value of character-repertoire in that array element, the value of font in that array element is set to the explicit modal default for font for the value of character-repertoire in the array element.

If the attribute-value is "null" for an attribute, or for one or more subattributes in the case of the emphasis attribute, and the attribute-extent is "global", the operation does not alter the corresponding attribute or subattribute values in any array element.

If attribute-id has value "emphasis", attribute-value may cause a total update to the emphasis attribute or a partial update when this capability exists, see 13.2 and clause B.17, in updated array elements and/or in the modal or global attribute values, see c).

The ATTRIBUTE operation does not change the display pointer.

NOTE – The note to 19.4.1.2 applies.

19.4.1.4 ERASE operation -

This operation has the form

ERASE erase-extent reset-attribute

It cancels the assignment of primary attribute value for some or all array elements, according to the value of erase-extent, and can also affect the values of secondary attributes in the same array elements, according to the value of reset-attribute.

The arguments are:

a) **erase-extent** specifies any valid address-extent as defined in 19.1.1.4 or 19.1.2.5;

b) **reset-attribute** takes a value from the set ("yes", "no"). If it takes value "yes", the secondary attributes of all the array elements affected by the ERASE operation are reset to their explicit modal default values, see 18.2. If it takes value "no", the secondary attribute values are unchanged.

The ERASE operation does not change the display pointer.

If ripple is enabled (19.2), the ERASE operation is replaced by a backward ripple performed with ripple coordinate x, the number of units equal to the number of array elements in the erase-extent and ripple-extent derived from the erase-extent as specified in 19.2.1.

19.4.1.5 Special operations for block-definition-capability

The following operations are available if blocks are in use for the DO and are subject to access control as defined in 19.5.

19.4.1.5.1 CREATE-BLOCK Operation

This operation has the form

CREATE-BLOCK Z B X_o Y_o D_x D_y

The Z and B arguments give the address at which the block is to be created. If a block currently exists at this value of (b,z), it will be deleted and the requested new block will replace it.

X_o and Y_o specify the origin of the block, used as the base for the p and q addressing within the block, see 13.1.2. X_o and Y_o are coordinate values on the X- and Y-dimensions of the display object. They are positive integers.

D_x and D_y specify the size of the block; neither can take a value which would cause the block to overflow the bounds of the Y-array in either X or Y dimension. They are positive integers.

This operation makes no changes to the contents of any array elements now contained in the new block.

19.4.1.5.2 DELETE-BLOCK operation

This operation has the form

DELETE-BLOCK Z B

The Z and B arguments give the address of the block to be deleted. If a block does not currently exist at this value of (b,z) the operation has no effect.

This operation makes no changes to the contents of any array elements contained in the block before deletion.

19.4.1.6 INSERT-X-ARRAY operation

This operation is available when ripple is enabled for the DO (see 19.2), and has the form

INSERT-X-ARRAY no-of-arrays

A forward ripple operation is performed, with ripple coordinate y and the number of units equal to no-of-arrays. The ripple-extent is derived as defined in 19.2.1 from an operation extent consisting of no-of-arrays x-arrays (or p-arrays).

If the backwards-forwards element of the RMCO has value "backwards", the x-array (or p-array) containing the display pointer is the first unit of the operation extent;

If backwards-forwards has value "forwards", the x-array (or p-array) following that containing the display pointer is the first unit of the operation extent.

NOTE – If blocks are not in use, the operation extent may cross more than one y-array.

Each array element of the first no-of-arrays x-arrays (or p-arrays) in the ripple-extent is filled as defined in 19.2.2.

If blocks are in use, INSERT-X-ARRAY is only permitted if the ripple-extent lies within one block. In particular, ripple-limit must be "p" or "q".

It is an error if any element of the ripple-extent is below any update-window.

If the backwards-forwards element of the RMCO has value "backwards", INSERT-X-ARRAY leaves the display pointer unchanged. If the backwards-forwards element of the RMCO has value "forwards", INSERT-X-ARRAY updates the y and z coordinates of the display pointer to point to the last x-array of the operation-extent, leaving the x coordinate unchanged.

19.4.1.7 DELETE-X-ARRAY operation

This operation is available when ripple is enabled for the DO (see 19.2), and has the form

DELETE-X-ARRAY no-of-arrays

A backward ripple operation is performed, with ripple coordinate y and the number of units equal to no-of-arrays. The ripple-extent is derived as defined in 19.2.1 from an operation extent consisting of no-of-arrays x-arrays (or p-arrays).

If the backwards-forwards element of the RMCO has value "backwards", the x-array (or p-array) containing the display pointer is the first unit of the operation extent.

If backwards-forwards has value "forwards", the x-array (or p-array) containing the display pointer is the last unit of the operation extent. The operation is only defined if there are at least (no-of-arrays – 1) x-arrays (or p-arrays) in the display object before that containing the display pointer.

NOTE – If blocks are not in use, the operation extent may cross more than one y-array.

If blocks are in use, DELETE-X-ARRAY is only permitted if the ripple-extent lies within one block. In particular, ripple-limit must be "p" or "q".

It is an error if any element of the ripple-extent is below any update-window.

If the backwards-forwards element of the RMCO has value "backwards", DELETE-X-ARRAY leaves the display pointer unchanged. If the backwards-forwards element of the RMCO has value "forwards", DELETE-X-ARRAY updates the y and z coordinates of the display pointer to point to the x-array immediately before the operation-extent, leaving the x coordinate unchanged.

19.4.1.8 INSERT-Y-ARRAY operation

This operation is available when ripple is enabled for the DO and blocks are not in use, and has the form

INSERT-Y-ARRAY no-of-arrays

A forward ripple operation is performed with ripple coordinate z and the number of units equal to no-of-arrays. The ripple-extent is derived as defined in 19.2.1 from an operation extent consisting of no-of-arrays y-arrays.

If the backwards-forwards element of the RMCO has value "backwards", the y-array containing the display pointer is the first unit of the operation extent;

If backwards-forwards has value "forwards", the y-array following that containing the display pointer is the first unit of the operation extent.

Each array element of the first no-of-arrays y-arrays in the ripple-extent is filled as defined in 19.2.2.

It is an error if any element of the ripple-extent is below any update-window.

If the backwards-forwards element of the RMCO has value "backwards", INSERT-Y-ARRAY leaves the display pointer unchanged. If the backwards-forwards element of the RMCO has value "forwards", INSERT-Y-ARRAY updates the z coordinate of the display pointer to point to the last y-array of the operation-extent, leaving the x and y coordinates unchanged.

19.4.1.9 DELETE-Y-ARRAY operation

This operation is available when ripple is enabled for the DO and blocks are not in use, and has the form

DELETE-Y-ARRAY no-of-arrays

A backward ripple operation is performed with ripple coordinate z and the number of units equal to no-of-arrays. The ripple-extent is derived as defined in 19.2.1 from an operation extent consisting of no-of-arrays y-arrays.

If the backwards-forwards element of the RMCO has value "backwards", the y-array containing the display pointer is the first unit of the operation extent.

If backwards-forwards has value "forwards", the y-array containing the display pointer is the last unit of the operation extent. The operation is only defined if there are at least (no-of-arrays – 1) y-arrays (or q-arrays) in the display object before that containing the display pointer.

It is an error if any element of the ripple-extent is below any update-window.

If the backwards-forwards element of the RMCO has value "backwards", DELETE-Y-ARRAY leaves the display pointer unchanged. If the backwards-forwards element of the RMCO has value "forwards", DELETE-Y-ARRAY updates the z coordinate of the display pointer to point to the y-array immediately before the operation-extent, leaving the x and y coordinates unchanged.

19.4.1.10 Copy operations

Operations are defined to copy the contents of a region of a display object to or from an external buffer. This buffer can be either a (named) RIO record or a temporary buffer. Copy operations are only available if the Ripple functional unit is selected but are not available when blocks are in use. Their use is independent of the value of *ripple-capability* VTE-parameter.

19.4.1.10.1 Copy buffer

A copy buffer stores the content and structure of an extent of the display object to allow subsequent transfer of this structure and content to some other extent of the DO.

A copy buffer may be held either in a RIO record (this choice is available only if the RIO functional unit is selected and a RIO has been included in the VTE), or in a special temporary buffer. There is one temporary buffer associated with each DO. Each temporary buffer is empty when a new full-VTE is established. The content of a temporary buffer is only changed by COPY-TO-BUFFER and COPY-LOGICAL-TO-BUFFER.

The content of the copy buffer is a sequence of DO updates taken from the set:

- TEXT
- ATTRIBUTE
- NEXT X-ARRAY
- NEXT Y-ARRAY
- POINTER-RELATIVE
- LOGICAL-TEXT
- LOGICAL-ATTRIBUTE
- NEXT FIELD
- LOGICAL-RELATIVE

NOTES

1 Each ATTRIBUTE or LOGICAL-ATTRIBUTE operation applies to exactly one array element.

2 The above provides a formal definition of the copy buffer. Typical implementations will use some other more efficient encoding with the same semantics, for example by grouping TEXT and ATTRIBUTE operations.

3 The single temporary buffer for a DO is also used by the operations defined in 19.4.2.5.

4 This International Standard does not specify or provide means of negotiating the size of the temporary buffer(s), but the Exception service, see clause 35, provides a means of advising overflow.

19.4.1.10.2 COPY-TO-BUFFER operation

This operation takes the form

COPY-TO-BUFFER end-address buffer-name
rendition structure

The arguments are:

a) **end-address** is a display pointer value greater than or equal to "current". It is either an explicit value or one of the symbolic values defined in 19.1.1.4 as valid as an extent end;

b) **buffer-name** is one of:

- a pair <RIO-name, record-id>, identifying a RIO record as in 22.2. RIO-name is optional if the VTE contains only one RIO;
- the symbolic value "temporary", identifying the temporary buffer associated with the DO;

c) **rendition** takes one of the values "copy attributes", "no attribute copy";

d) **structure** takes one of the values "none", "x", "x and y".

The operation initialises then fills in the copy buffer designated by buffer name. A copy buffer is initialised by creating it (if it is a RIO record which does not exist) followed by setting its contents to the empty sequence.

The copy operation applies to the set of array elements (x, y, z) which satisfy

$x_c \le x \le x_f$; $y_c \le y \le y_f$; $z_c \le z \le z_f$
(if *structure* is "x and y")

$x_c \le x \le x_f$; $(y_c , z_c) \le (y , z) \le (y_f , z_f)$
(if *structure* is "x")

$(x_c , y_c , z_c) \le (x , y , z) \le (x_f , y_f , z_f)$
(if *structure* is "none")

where (x_c , y_c , z_c) is the current value of the display pointer and (x_f , y_f , z_f) is *end-address*.

The selected elements are processed in order of address, as follows. For each array element, DO updates are added in order to the end of the current contents of the buffer:

a) for each element after the first, if the element is in a different x-array or y-array, add DO updates as specified in table 8.

Table 8 Copy insertion

structure	new Y-array?	new X-array?	add
x&y	Y	–	NEXT-Y-ARRAY
x&y	N	–	NEXT-X-ARRAY
x	Y	–	NEXT-X-ARRAY
x	N	Y	NEXT-X-ARRAY

b) if *rendition* = "copy attributes", for each secondary attribute which has a value assigned, add

ATTRIBUTE attribute-id attribute-value
<"current","current">

If both character-repertoire and font have assigned values, character-repertoire must be processed before font.

c) if the primary attribute has a value assigned, add

TEXT primary-attribute-value

else if the primary attribute has no value assigned, add

POINTER-RELATIVE m=1 (i.e., x := x + 1)

The COPY-TO-BUFFER operation does not affect any DO content or any display pointer.

19.4.1.10.3 COPY-FROM-BUFFER operation

This operation takes the form

COPY-FROM-BUFFER start-address buffer-name
rendition structure ripple

The arguments are:

a) **start-address** is either an explicit value for the display pointer or is one of the symbolic values defined in 19.1.1.4 which is valid as an extent start;

b) **buffer-name**, **rendition** and **structure** are defined as for the corresponding parameters in COPY-TO-BUFFER in 19.4.1.10.2;

c) **ripple** takes one of the values "on", "off". The value is ignored if ripple is not enabled.

The operation proceeds as follows:

- set the display pointer to *start-address*;
- obey the sequence of DO updates in the copy buffer designated by *buffer-name*.

In obeying the DO updates, the following interpretations apply.

Let (x_s, y_s, z_s) be the *start-address*.

- if *structure* is "none", NEXT-X-ARRAY is ignored , otherwise, it causes y := y + 1 or, if y=y-bound, NEXT-Y-ARRAY, and x := x_s;
- if *structure* is "none", NEXT-Y-ARRAY is ignored; if *structure* is "x", NEXT-Y-ARRAY causes y := y + 1 or, if y=y-bound, NEXT-Y-ARRAY and x := x_s; if *structure* is "x and y", NEXT-Y-ARRAY causes z := z + 1; y := y_s and x := x_s;
- ATTRIBUTE is ignored if *rendition* = "no attribute copy" and otherwise processed as below.

If *ripple* argument is "off" (or ripple is not enabled for the DO), ATTRIBUTE, TEXT and POINTER-RELATIVE are processed as if ripple-mode in the RMCO were set to "false".

If *ripple* is "on", ATTRIBUTE, TEXT and POINTER-RELATIVE updates are processed in groups. Each group consists of zero or more ATTRIBUTE updates, followed by either TEXT or POINTER-RELATIVE. For each group, a forward ripple occurs with ripple of one unit, then the ATTRIBUTE, TEXT and POINTER-RELATIVE updates are processed as if ripple-mode in the RMCO were set to "false". POINTER-RELATIVE is interpreted as meaning "move to the next element of the ripple-extent". The ripple-extent is based on an operation extent consisting of the single array element at the display pointer. The ripple coordinate is always x.

The COPY-FROM-BUFFER operation is invalid if any of the individual operations generated are invalid, for example because bounds are violated.

A COPY-FROM-BUFFER from a RIO record is only defined if the RIO record content was created by a COPY-TO-BUFFER operation.

The COPY-FROM-BUFFER operation is invalid if the copy buffer contains any logical pointer operations.

No change is made to the source copy buffer.

NOTE – clause 22 defines other operations on RIOs which are independent of the above copy operations.

19.4.2 Update operations using logical pointer

This sub-clause defines the update operations which are available for updating the content of array elements of a display object when fields are in use for a DO. All these operations are subject to access control as defined in 19.5.

NOTE – See note 1 in 13.1.3.

19.4.2.1 LOGICAL-TEXT operation

This operation has the form

LOGICAL-TEXT primary-attribute-value FDR-attributes

The arguments are:

a) **primary-attribute-value**: as for TEXT operation in 19.4.1.1;

b) **FDR-attributes**: takes a value from the set ("yes","no"). If it takes value "yes" then the secondary attributes of the updated array element are given values as taken from the FDR field attributes, see 20.3.3.3, as first choice, see below. If it takes value "no", then the field modal attributes are used as the first choice.

Where the 'first choice' as above is "null" for an attribute, the rules in b), c), e), f) and g) in 19.4.1.1 apply.

If ripple is enabled, a forward ripple with ripple of one unit is performed before the primary and secondary attributes are updated. The ripple-extent is defined as specified in 19.3.1 from the extent consisting of the single array element at the logical pointer.

19.4.2.2 REPEAT-LOGICAL-TEXT operation

This operation has the form

REPEAT-LOGICAL-TEXT finish-address
FDR-attributes primary-attribute-value-string

It has the effect of repeating a LOGICAL-TEXT operation (19.4.2.1) for array elements within the repeat extent as defined below.

The arguments are:

a) **finish-address** is combined with "log-current" to produce repeat-extent; it can take a value as in 19.1.3.5 greater than or equal to "log-current". If not all characters in primaryattribute-value-string are exhausted, the remaining characters are lost;

b) **FDR-attributes**: as defined in 19.4.2.1;

c) **primary-attribute-value-string** is as defined in 19.4.1.2 and is used as defined therein.

After a REPEAT-LOGICAL-TEXT operation, the logical pointer identifies the position immediately after the last array element updated.

If ripple is enabled, a forward ripple is performed before the REPEAT-LOGICAL-TEXT operation, with the number of units equal to the number of array elements in the repeat-extent. The ripple-extent is derived from the repeat-extent as specified in 19.3.1.

19.4.2.3 LOGICAL-ATTRIBUTE operation

This operation has the form

LOGICAL-ATTRIBUTE attribute-id attribute-value attribute-extent

where **attribute-id** is as defined in 19.4.1.3 and **attribute-value** is as there defined but with an additional value "field-explicit-value" available; if this value is used then the attribute value is taken from the FDR field attributes for each field to which the ATTRIBUTE operation applies, see attribute-extent below or, if the FDR value is "null", then the explicit modal default value is used, see 18.2.

Attribute-extent takes one of the following forms:

a) any valid logical address extent as defined in 19.1.3.5;

b) "global": this extent applies to all fields on all Y-arrays, but does not apply to array elements outside fields;

c) "modal": causes the field modal attribute value for the field currently addressed by the logical pointer to be set to attribute-value, see also 13.1.3.2. Other information on "modal" extent in 19.4.1.3 applies here.

The action of LOGICAL-ATTRIBUTE operation with respect to character-repertoire and font secondary attributes is as defined in 19.4.1.3.

The LOGICAL-ATTRIBUTE operation does not change the logical pointer.

19.4.2.4 LOGICAL-ERASE operation

This operation is available when field-definition-capability is selected for a DO.

This operation has the form

LOGICAL-ERASE logical-erase-extent reset-attribute

It cancels the assignment of primary attribute value for some or all array elements in fields, according to the value of logical-erase-extent, and can also affect the values of secondary attributes in the same array elements, according to the value of reset-attribute.

The arguments are

a) **logical-erase-extent** takes any valid logical address extent as defined in 19.1.3.5;

b) **reset-attribute** takes a value from the set ("yes", "no"). If it takes value "yes", the secondary attributes of each array element affected by the LOGICAL-ERASE operation are reset to the field attribute values assigned by the FDR for the appropriate field or, for an attribute for which no such value is assigned, to the explicit modal default value. If it takes value "no", the secondary attribute values are unchanged.

The LOGICAL-ERASE operation does not change the logical pointer.

If ripple is enabled, the LOGICAL-ERASE operation is replaced by a backward ripple performed with the number of units equal to the number of array elements in the logical-erase-extent and with ripple-extent derived from the logical-erase-extent as specified in 19.3.1.

19.4.2.5 Logical copy operations

The following operations are equivalent to those defined in 19.4.1.10 and in addition to conditions stated there are available only if fields are in use for the DO. They share the same temporary buffer. RIO-records may also be used.

19.4.2.5.1 COPY-LOGICAL-TO-BUFFER operation

This operation takes the form

COPY-LOGICAL-TO-BUFFER end-address buffer-name rendition structure

The arguments are:

a) **end-address** is a logical pointer value greater than or equal to log-current. It is either an explicit value or one of the symbolic values defined in 19.1.3.5 as valid as an extent end;

b) **buffer-name** is one of:

- a pair <RIO-name, record-id>, identifying a RIO record as in 22.2. RIO-name is optional if the VTE contains only one RIO;
- the symbolic value "temporary", identifying the temporary buffer associated with the DO;

c) **rendition** takes one of the values "copy attributes", "no attribute copy";

d) **structure** takes one of the values "none", "k".

The operation initialises and then fills in the copy buffer designated by buffer-name. A copy buffer is initialised by creating it (if it is a RIO record which does not exist) followed by setting its contents to the empty sequence.

The copy operation applies to the set of array elements (k, f, z) which lie within active fields and which satisfy

$k_c \le k \le k_f$; $(f_c , z_c) \le (f , z) \le (f_f , z_f)$ (if structure value is "k")

$(k_c , f_c , z_c) \le (k , f , z) \le (k_f , f_f , z_f)$ (if structure value is "none")

where (k_c , f_c , z_c) is the current value of the logical pointer and (k_f , f_f , z_f) is the end address.

The selected elements are processed in order of address, as follows. For each array element, DO updates are added in order to the end of the current contents of the buffer:

- if structure = "k" and this is not the first element to be processed and this element is in a different field from the previous element then add NEXT-FIELD;
- if rendition = "copy-attributes" then, for each secondary attribute which has a value assigned add LOGICAL-ATTRIBUTE attribute-id attribute-value <"log-current", "log-current">. Attribute-value shall not take the value "field-explicit-value". If both character-repertoire and font have assigned values, repertoire must be processed before font;
- if the primary attribute has a value assigned, add LOGICAL-TEXT primary-attribute-value FDR-attribute = "no" else add LOGICAL-RELATIVE u=1 (i.e., k:=k+1).

The COPY-LOGICAL-TO-BUFFER operation does not affect any DO content or the logical pointer.

19.4.2.5.2 COPY-LOGICAL-FROM-BUFFER operation

This operation takes the form

COPY-LOGICAL-FROM-BUFFER start-address
buffer-name rendition structure ripple

The arguments are:

a) **start-address** is an explicit value or one of the symbolic values defined in 19.1.3.5 as valid as an extent start;

b) **buffer-name** is one of:
- a pair <RIO-name, record-id>, identifying a RIO record as in 22.2. RIO-name is optional if the VTE contains only one RIO,
- the symbolic value "temporary", identifying the temporary buffer associated with the DO;

c) **rendition** takes one of the values "copy attributes", "no attribute copy";

d) **structure** takes one of the values "none", "k";

e) **ripple** takes one of the values "on", "off". The value is ignored if ripple is not enabled.

The operation proceeds as follows:
- set the logical pointer to start-address;
- obey the sequence of DO updates in the copy buffer designated by buffer-name.

In obeying the DO updates, the following interpretations apply:

Let (k_s,f_s,z_s) be the start address.
- NEXT-FIELD is ignored if structure = "none". If *structure* = "k" and there is another active field in the forward f direction, f is incremented to a value where the next active field exists on the current y-array. If such a field does not exist on the current y-array, z:=z+1, f:=1. This process is repeated until an active field is reached. Then $k:=k_s$. The operation terminates if no further active fields can be found;
- LOGICAL-ATTRIBUTE is ignored if *rendition* = "no attribute copy" and otherwise processed as below.

If *ripple* argument is "off" (or ripple is not enabled for the DO), LOGICAL-ATTRIBUTE, LOGICAL-TEXT and LOGICAL-RELATIVE are processed as if ripple-mode in the RMCO were set to "false".

If *ripple* is "on", LOGICAL-ATTRIBUTE, LOGICAL-TEXT and LOGICAL-RELATIVE updates are processed in groups. Each group consists of zero or more LOGICAL-ATTRIBUTE updates, followed by either LOGICAL-TEXT or LOGICAL-RELATIVE. For each group, a forward ripple occurs with a ripple of one unit, then the LOGICAL-ATTRIBUTE, LOGICAL-TEXT and LOGICAL-RELATIVE updates are processed as if ripple-mode in the RMCO were set to "false". LOGICAL-RELATIVE is interpreted as "move to the next element in the ripple-extent".

The logical-ripple-extent is based upon an operation extent consisting of the single array element at the logical pointer.

The COPY-LOGICAL-FROM-BUFFER operation is invalid if any of the individual operations generated are invalid, for example because bounds are violated.

A COPY-LOGICAL-FROM-BUFFER from a RIO record is only defined if the RIO record content was created by a COPY-LOGICAL-TO-BUFFER operation.

The COPY-LOGICAL-FROM-BUFFER operation is invalid if the copy buffer contains any primitive pointer operations.

No change is made to the source copy buffer.

NOTE – clause 22 defines other operations on RIOs.

19.5 Access control over display object

19.5.1 Access control for operations other than logical operations

19.5.1.1 Display pointer and array element update operations

All addressing operations (updating primitive or extended display pointer, see 13.1.1.1) and TEXT, REPEAT TEXT, ATTRIBUTE, and ERASE operations are subject to the following conditions:

a) access-rule for display object is satisfied;

and

b) at least one of
1) fields are not in use,
2) access-outside-fields = "allowed",
3) VT-user is not designated as the Terminal VT-user, see 19.5.2.2.

19.5.1.2 CREATE-BLOCK and DELETE-BLOCK operations

These operations are subject to the access control in 19.5.1.1 and in addition are only available if blocks are in use for the DO.

19.5.2 Asymmetry of VT-users

Use of the FDCO control object, see 14.2 and 20.3.3, is closely associated with use of fields. This CO has an access-rule, applied by *CO-access* VTE-parameter, which has effects as defined below. Use of the FEICO and FEPCO control objects for controlled data entry is optional, see 14.2 and 20.3.3.6, but if used there is an additional affect on display object access control as defined below.

In S-mode if fields are in use for the DO an asymmetry between the two VT-users may be defined by the access-rule for the FDCO (see 14.2 and 20.3.3). If the FDCO access-rule is WAVAR & WACI, the VT-user which initiated the VT-association is designated as the **Application VT-user** and the other VT-user is designated as the **Terminal VT-user**. If the FDCO access-rule is WAVAR & WACA, the VT-user which accepted the VT-association is designated as the **Application VT-user** and the other VT-user is designated as the **Terminal VT-user**. If the FDCO has any other access rule (and always in A-mode) neither VT-user is designated as an Application VT-user or a Terminal VT-user.

19.5.2.1 No designation of asymmetry

Where no designation of asymmetry is made, all DO operations are subject to the DO-access rule. In addition, logical update operations (19.4.2) may only be used to update active fields (20.3.3).

19.5.2.2 Designation of asymmetry

Where designation of asymmetry is made, all DO operations are subject to the DO-access rule. In addition, logical update operations (19.4.2) may only be used to update active fields (20.3.3.5).

A Terminal VT-user may be further restricted if a FEICO is used. If the entry-control-list in the FDR for a particular field is not "void", see 20.3.3.6 and B.18.4, then the Terminal VT-user is subject to any constraints applied by the list of entry-controls, see B.18.4 and B.18.7 (the entry-controls may be such as to apply no additional constraint).

Access by a Terminal VT-user to other DO operations is covered in 19.5.1.

20 Control object VTE-parameters

Table 9 defines in part 1 the VTE-parameters which apply to a CO; each CO requires one set of these parameters. Some VTE-parameters or parameter values are dependent on the selection of functional units as indicated in the table.

When *CO-structure* takes a value other than "non-parametric", then, in addition to the VTE-parameters defined in part 1 of table 9, a set of the VTE-parameters defined in part 2 of table 9 applies for each data element.

Table 9 - Control Object VTE-parameters

Part 1 – Control object common VTE-parameters

Parameter	Value
CO-name	a character string of type ASN.1 PrintableString
CO-type-identifier	either a value of ASN.1 OBJECT IDENTIFIER type or a character string of type ASN.1 PrintableString
CO-structure	"non-parametric", number of data elements 1...N; (default = 1); see note 3
CO-access	"NSAC", "WAVAR", "WACI", "WACA", "WAVAR & WACI"*, "WAVAR & WACA"*, "no-access"*, (default = "NSAC"); see note 1
CO-priority	"normal", "high", "urgent" (default = "normal"); see note 2
CO-trigger	optional: "not selected", "selected" (default = "not selected")

NOTES
1 Values for CO-access marked * are valid only if Enhanced Access Rules functional unit is selected.

2 A CO for which *CO-priority* "normal" is invalid, see 20.1.4, will be taken as of "high" priority in the default case.

3 This parameter is used only if the Structured Control Objects functional unit is selected; the default value applies if the parameter is not present.

Part 2 – Control object data element VTE-parameters

Parameter	Value
CO-element-id	conditional: integer: required if the number of data elements is greater than one; see note 4
CO-category	"character", "boolean", "symbolic", "integer", "transparent" (default = "boolean")
CO-repertoire-assignment	this VTE-parameter has the same form and default as *repertoire-assignment* in 18.2.4. This VTE-parameter is relevant to the control object only if *CO-category* takes value "character"
CO-size	data storage capacity of data element

NOTE
4 This parameter is used only if the Structured COs functional unit is selected; the default value applies if the parameter is not present.

20.1 Usage and effects of control object VTE-parameters

The following additional information applies to the use and effects of the VTE-parameters listed in table 9.

20.1.1 CO-type-identifier specifies the source of semantic definition for the control object. When it is of type ASN.1 OBJECT IDENTIFIER, table 10 gives permitted values and corresponding definition sources.

If *CO-type-identifier* has a character string value, then the value of this string and any semantics of the information field content are established by means outside the scope of this International Standard.

Table 10 - OBJECT IDENTIFIER values for control objects

Control object generic type	CO-type-identifier values
Control object type defined in this International Standard	See C.1.1.1
Registered control object type	See C.1.1.2

20.1.2 If **CO-structure** takes the value "non-parametric", the information structure of the control object and its initial value is defined as identified by the value of *CO-type-identifier*; the structure definition will then also identify whether any partial updating of the control object information field is possible.

If **CO-structure** takes an integer value, then the information structure is defined parametrically and the value defines the number of data elements in the information structure; each such data element is defined by a set of the parameters listed in part 2 of table 9 and can be individually addressed to effect partial updating of the control object information field.

20.1.3 CO-access specifies the access-rule for the control object, and determines whether and when each VT-user may update the object, see clause 9. The value "no-access" implies that the information content of the CO cannot be altered or extended and has to be defined in a register entry or profile. Table 11 specifies the permitted combinations of *CO-access*, *CO-priority* and *VT-mode*.

NOTE – VT-users defining control objects with CO-access "NSAC" should be aware that collisions may occur with such control objects. In actual fact, due to the way in which control objects are generally used (the semantics of the control object), collisions are unlikely, but VT-users should be aware of the fact that the service provider does nothing to prevent or resolve them.

20.1.4 CO-priority assigns to the control object a value for update priority. This VTE-parameter controls the handling of updates to the control object in relation to updates to other objects, as defined in 24.5. Table 11 defines permitted values in relation to values of *CO-access*.

Control objects with *CO-priority*="urgent" are used to support the non-destructive interrupt facility, see 7.7 and B.16.2.

NOTES

1 Implementations will typically place tight limits on the permitted combinations of *CO-name*, *CO-category* and *CO-size* if *CO-priority* has value "urgent", see annex D;

2 There is no relation between the selection of the Urgent Data functional unit and CO VTE-parameter *CO-priority* = "urgent".

20.1.5 The effect of **CO-trigger** is defined in 24.1 and 31.1.4. CO-trigger may only be "selected" if *CO-priority* is "normal".

Table 11 - Permitted combinations of VT-mode, CO-access and CO-priority

VT-mode &	**S-MODE**			**A-MODE**		
CO-priority	normal	high	Urgent	normal	high	Urgent
CO-access						
WACI	n/a	NDC *	NDC *	DC	NDC	NDC
WACA	n/a	NDC *	NDC *	DC	NDC	NDC
WAVAR	DC	NDC	NDC	n/a	n/a	n/a
WAVAR-&-WACI	DC *	NDC *	NDC *	n/a	n/a	n/a
WAVAR-&-WACA	DC *	NDC *	NDC *	n/a	n/a	n/a
NSAC	n/a	NDC	NDC	DC	NDC	NDC
no-access	VALUES HAVE NO EFFECT					

Key:
* – only applies if Enhanced Access-rules functional unit is selected; otherwise, n/a
n/a – combination not permitted
DC – subject to delivery control, if any; can have trigger
NDC – not subject to delivery control; cannot have trigger

20.2 Usage and effects of data element VTE-parameters

The following additional information applies to the use and effects of the control object data element VTE-parameters listed in part 2 of table 9. These VTE-parameters apply to each individual data element (if the Structured Control Objects functional unit is not selected, this is still true, but there can only be one data element and *CO-element-id* is not used).

20.2.1 **CO-element-id** is used to identify a data element within the control object for the purposes of updating the element and of negotiating values of the other VTE-parameters defined for the element. For each data element taken in order from the first the value assigned to *CO-element-id* starts at 1 and increases by 1 until the last which takes the value of *CO-structure*.

NOTE – There are two levels of partial updating of a control object which has parametrically-defined structure:

a) each data element can be individually updated;

b) if a data element has *CO-category* "boolean" then its constituent booleans can each be individually updated.

20.2.2 The appropriate value for **CO-size** depends on the value of *CO-category* as follows:

a) character: maximum length of character string (default = 16 characters);

b) boolean: maximum number of boolean values (default = 16);

c) symbolic: maximum number of distinct values (default = 256);

d) integer: maximum value of integer (default = 65535);

e) transparent: maximum number of bits (default = 16 bits).

20.2.3 Except for boolean category elements, a control object data element contains a single unit of information (i.e., value) of the type specified by *CO-category*, i.e., a single character-string, symbolic value, integer, or transparent (uninterpreted) bit string. When a new value is entered into a control object of one of these CO-categories, it completely replaces the old value.

When a CO data element has *CO-category* value "boolean", the element may contain one or more boolean values. Each boolean is updated individually without affecting the other booleans contained in the control object. Each boolean is addressed by its numeric order relative to the other booleans contained in the control object, i.e., a boolean is addressed as "Boolean n" where n is in the range $1 \leq n \leq$ CO-size.

Using this numbering convention a value for a boolean in such a control object is given as one of the following

n."true" or n."false", where n is the boolean number.

20.2.4 Except where specified otherwise in this International Standard or in an external CO definition (e.g., in a register), the initial value for each control object data element when a VTE becomes the current-VTE depends on the *CO-category* value as follows:

a) character: a "null" zero-length string;

b) boolean: each boolean has value "false";

c) symbolic: "null";

d) integer: 0;

e) transparent: each defined bit has value 0.

20.2.5 When *CO-category* = "character", all characters in the string have the same repertoire as specified by *CO-repertoire-assignment* or its default. The value of *CO-repertoire-assignment* is set through negotiation, in a VTE-profile or in a registered control object.

20.2.6 When *CO-category* takes the value "symbolic" in the CO specification, each symbolic value is assigned a distinct integer value in the range 0 to *CO-size* – 1. The special value "null" shall always be assigned the value 0.

20.3 Standard control objects

20.3.1 Termination Control Object (TCO)

A TCO can be used with a device object, usually in conjunction with the device object termination conditions, see 23.4.

To define a TCO, the control object VTE-parameters take the following values:

CO-name: any value unique to the VTE;
CO-type-identifier: ASN.1 OBJECT IDENTIFIER value vt-b-sco-tco, see annex C;
CO-access: any valid value except "NSAC"; see 23.4;
CO-priority: "normal";
CO-trigger: any valid value;
CO-category: "integer";
CO-size: takes default value.

The initial value of a TCO is 0 (zero). See 23.4 for a description of use of this CO.

20.3.2 Echo Control Object (ECO)

For A-mode, an optional control object is defined to allow a VT-user to control the enabling of echo by the peer VT-user. When the VT-users negotiate the use of this control object, the values of the VTE-parameters for this control object are as defined as follows:

CO-name: E
CO-type-identifier: ASN.1 OBJECT IDENTIFIER value vt-b-sco-echo, see annex C;
CO-access: "WACA", "WACI" or "NSAC"; see Note 2
CO-priority: "normal"
CO-trigger: "selected" or "not-selected" (default = "selected")
CO-category: "boolean"
CO-size: 1 boolean

The initial value of the single boolean in the ECO control object is "false".

When fields are in use for either DO, the function of the echo control object is superseded by the facilities of the FEICO and FEPCO, see 20.3.4 and 20.3.5. The ECHO control object can exist in a VTE, but has no effect.

See clauses B.5, B.6 and B.7 for notes on use of this control object.

NOTES

1 If no echo control object exists, it is the local VT-user's concern whether input updates are echoed locally or not.

2 A value of "NSAC" for CO-access should be used when and only when it is legitimate for either VT-user to alter the state of echo control. The possibility of collisions must be taken into account.

20.3.3 Field Definition Control Object (FDCO)

20.3.3.1 VTE-parameters for FDCO

To define a FDCO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;

CO-type-identifier: ASN.1 OBJECT IDENTIFIER value vt-b-sco-fdco, see annex C, identifying the CO as being of type FDCO;

CO-structure: "non-parametric", meaning that the information structure is formally referenced by the value of CO-type-identifier; it is fully defined by this International Standard, see 20.3.3.2;

CO-access: S-mode: "WAVAR", "WAVAR & WACI", "WAVAR & WACA" or "no-access" (see Note 2); A-mode: same as associated DO or "no-access";

CO-priority: "normal";

CO-trigger: "not selected".

NOTES

1 20.3.3.5 gives information on the use of the FDCO in using Fields.

2 The value of CO-access for FDCO has an effect on access control over the display object as defined in 19.5.2. A note on further implicit semantics is given in clause 12.

20.3.3.2 Definition of information content for FDCO

The content of a FDCO is an array of records known as FDRs. These records are identified for generation and update purposes by the z and f coordinates of the field they define, see 13.1.3

Each FDR is itself composed of a number of items as defined in 20.3.3.3, known as FDR-components.

For the purpose of update of a FDCO, the VT-object-data service sub-parameter, see 31.1.3.1, takes the form

Z F sequence of
<FDR-component-id FDR-component-value>

where Z and F identify the FDR and the FDR-components and possible values are as defined in 20.3.3.3.

If the VTE-parameter *dimensions* = 2, the Z component shall be omitted. If *dimensions* = 3, the Z component is optional; if it is absent, then the Z value is taken from the z-coordinate of the current position of the logical pointer.

NOTE – The FDCO itself is identified by *VT-object-ident* service sub-parameter, see 31.1.3.1

20.3.3.3 Definition of content of FDR : FDR-components

The content of one FDR normally consists of the following FDR-components, some of which are themselves complex; when a field does not exist, i.e., field-status has value "non-extant", see a), the other FDR-components can be considered to have no further existence or to be given "void" values.

NOTE – If certain values of z are below the z-update-window, any FDRs relating to these values of z are conceptually not destroyed, but since any update of such FDRs can have no effect on the DO it is not necessary to retain their contents (except, perhaps, for local reasons).

a) **field-status**: takes a value from the set "active", "inactive", "non-extant", see 20.3.3.5;

b) **field-extent**: value is a sequence of 4-tuples <Xi Yi Dxi Dyi> where each 4-tuple defines a field-element, see 13.1.3. The order of the field-elements in field-extent defines the order of the field-elements for the purpose of evaluating the K-dimension for the field, see 13.1.3.1;

c) **field-attributes**: there is one component of the FDR corresponding to each secondary attribute. The value is either a valid value for the secondary attribute or is "null";

d) **next-field**: value is an integer taken as a value of f coordinate or can be "void", see 20.3.3.4;

e) **previous-field**: value is an integer taken as a value of f coordinate or can be "void", see 20.3.3.4;

f) **T-policy**: is the transmission policy for the field and takes one of the following values :
- transmit all of this field;
- transmit all of this field if any portion of it has been changed;
- transmit those portions of this field, if any, which have been changed;
- do not transmit any of this field;
- use global value of transmission policy, i.e., as given in TPCO, see 20.3.7;

g) **entry-control-list**: value is a sequence (possibly empty) of entry-controls, see 20.3.3.6. Each part of each entry-control (i.e., device-object-list, FEIR-list, FEPR-list) is a separate FDR component and may be separately updated.

20.3.3.4 Field Set Navigation Path

Where a set of fields is defined by a set of FDRs, the normal path for moving through the set of fields can be explicitly defined using *next-field* and *previous-field* FDR-components. This path is referred to as the 'navigation path'. A distinction is made between the forwards and backwards navigation paths; these can be different, i.e., the field visited going back from a particular field can be different from that from which this field was reached going forwards.

Each of these is an integer taken as a coordinate value on the F dimension.

If either has value "void", that part of the navigation path is implicit from the F dimension. These links affect the operation of NEXT-FIELD and PREVIOUS-FIELD operations on the logical pointer in 19.1.3.2.2.

If *next-field* or *previous-field* is not "void" but has a value at which no field is currently "active" or "inactive", i.e., the value of *field-status* is implicitly or explicitly "non-extant", the field is the last of the 'forwards' navigation path or of the 'backwards' navigation path respectively. The value 0, not normally a valid value for f coordinate, can be used here to force this condition.

Where the navigation path from a field is implicit from the F dimension (*next-field* and/or *previous-field* is "void"), a skip will occur to the next "active" or "inactive" field in the appropriate direction on F; if there is no such field the field is taken as the last of the path.

Where a field is encountered which is "inactive", the navigation path (forward or backward) will skip this field (whether implicit or explicit navigation is in force) and endeavour to find a further field from the next/previous value in this field. If no "active" field can be found, the field prior to the "inactive" field will be taken as the last of the path.

If a field is deleted (*field-status* becomes "non-extant"), any navigation paths to that field are broken; no memory is kept of the pointers from the deleted field and no automatic re-linking takes place.

NOTES

1 Since the z coordinate is not present in *next-field* or *previous-field*, the set of fields within which the path can be specified in this way is wholly contained within one Y-array.

2 To avoid the breaking of the navigation path when a field is deleted, the *field-status* can be set to "inactive".

20.3.3.5 Use of FDCO for Creation, Modification and Deletion of Fields

The status of a field is determined by the value of FDR-component *field-status* in the FDR. If the value is "active" or "inactive", the field exists; if it is "non-extant", the field does not exist.

The initial state of the FDCO, when a VTE is brought into use, is that all FDRs conceptually exist, with *field-status* "non-extant".

A field is **created** by updating its FDR (see 20.3.3.2) with required values and including *field-status* value "active" or "inactive" (other FDR-components cannot have non-void values if *field-status* value is "non-extant").

The characteristics of an existing field can be **modified** by updating the appropriate FDR-component values in its FDR, including a change of *field-status* between "active" and "inactive".

A field is **deleted** by updating its FDR with *field-status* "non-extant"; other FDR-components in the FDR are automatically set to "void" and cannot be recovered subsequently.

If any secondary attribute, including character-repertoire, is given an explicit value or value "null" in *field-attributes*, this value replaces any previous value (or "null") for this attribute in the FDR. Other attribute values in the FDR are unchanged by the FDR update.

20.3.3.6 Definition of entry-control-list in FDR

This is a sequence of one or more *entry-controls*, see below, or can be "void" (the empty sequence). The sequence is significant only in that it is indexed by the CCO value at termination, see 20.3.6.2. Entry-controls enable entry instructions and reactions to be applied to the field and associated with subsets of the device objects linked to the display object. If the value is "void", use of the field is controlled only by the access-rule for the display object; see also 19.5.

Each **entry-control** is a set of one each of the following parts, which can themselves be complex:

a) **device-object-list**: this is a sequence of device object names or can be "void" (the empty sequence). The sequence is significant only in that it is indexed by the CCO value at termination, see 20.3.6.2. Device objects included can enter data into the field subject to the FEIR-list. Value "void" implicitly includes all those device objects which are linked to the display object by VTE-parameter *device-display-object*.

   A device object can appear in the device-object-list in more than one entry-control. The combination of constraints or permissions, thus applied, is according to the definition in the FEICO register entries, see B.18.7.

   If there is any non-void device-object-list associated with a field, then any device object which is in no device-object-list for the field cannot enter data into the field.

   NOTE 1 – Device-object-list can include device objects which are not explicitly linked to the display object with which this FDCO is associated.

b) **FEIR-list**: this is a set of FEIR identifiers or can be "void" (the empty sequence). Each is a pair <FEICO-name, index> where FEICO-name is the name of a CO of type FEICO, see 20.3.4, and index is an integer addressing a record in this CO. Value "void" implies that there is no constraint on entry into the field from the device objects in device-object-list. If index does not identify an existing record in the FEICO or if the FEICO does not exist, the FEIR-list entry is ignored. If this is true for all entries, this is equivalent to value "void" for FEIR-list.

c) **FEPR-list**: this is a sequence of FEPR identifiers or can be "void" (the empty sequence). Each is a pair <FEPCO-name, index> where FEPCO-name is the name of a CO of type FEPCO, see 20.3.5, and index is an integer addressing a record in this CO. The sequence is significant only in that it is indexed by the CCO value at termination, see 20.3.6.2. If index does not identify an existing record in the FEPCO or if the FEPCO does not exist, the FEPR-list entry is ignored. If this is true for all entries, this is equivalent to value "void" for FEPR-list.

Value "void" for all parts of any one entry-control is equivalent to value "void" for entry-control-list.

NOTE 2 – The structure of entry-control-list is illustrated in B.18.14.

20.3.4 Field Entry Instruction Control Object (FEICO)

20.3.4.1 VTE-parameters for FEICO

To define a FEICO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;

CO-type-identifier: a value as in table 9 to select a register entry or private type definition which defines an information structure conforming to 20.3.4.2;

CO-structure: "non-parametric";

CO-access: takes either the same value as *CO-access* for the FDCO associated with the same display object, see 23.1 and 23.5, or the value "no-access";

CO-priority: takes value "normal";

CO-trigger: takes value "not-selected".

NOTE – The value "no-access" for *CO-access* implies that the FEICO is available only for reference to the FEIRs generated from the register entry. These FEIRs cannot be altered and further FEIRs cannot be inserted.

20.3.4.2 Definition of information content for FEICO

The content of a FEICO is an array of records known as Field Entry Instruction Records (FEIRs). This array is not explicitly bounded and new records can be generated using the CO update operation, see 31.1. A FEIR-index of type Integer is used to enable individual FEIRs to be addressed for update or for reference in a FDR, see 20.3.3.6.

For the purpose of update of a FEIR, the *VT-object-data* parameter, see 31.1, takes the form

FEIR-index FEIR-content-value.

The FEIR-content-value is not standardised in this International Standard but B.18.7 describes typical content. A register entry for a FEICO will define the permissible content for FEIRs in such a FEICO together with transfer syntax for generating a new FEIR where this is permitted by the register entry.

20.3.5 Field Entry Pilot Control Object (FEPCO)

20.3.5.1 VTE-parameters for FEPCO

To define a FEPCO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;

CO-type-identifier: a value as in table 9 to select a register entry or private type definition which defines an information structure conforming to 20.3.5.2;

CO-structure: takes the value "non-parametric";

CO-access: takes either the same value as CO-access for the FDCO associated with the same display object, see 23.1 and 23.5, or the value "no-access";

CO-priority: takes the value "normal";

CO-trigger: takes the value "not-selected".

NOTE – The value "no-access" for *CO-access* implies that the FEPCO is available only for reference to the FEPRs generated from the register entry. These FEPRs cannot be altered and further FEPRs cannot be inserted.

20.3.5.2 Definition of information content for FEPCO

The content of a FEPCO is an array of records known as Field Entry Pilot Records (FEPRs). This array is not explicitly bounded and new records can be generated using the CO update operation, see 31.1. A FEPR-index of type Integer is used to enable individual FEPRs to be addressed for update or for reference in a FDR, see 20.3.3.6.

For the purpose of update of a FEPR the *VT-object-data* service sub-parameter, see 31.1.3.1, takes the form

FEPR-index FEPR-content-value.

FEPR-content-value takes the form

Event Conditions Reactions

where

Event ::= CHOICE {...}

Conditions ::= SET OF { CHOICE {...} }

Reactions ::= SEQUENCE OF { CHOICE {...} }

This International Standard does not define the sets of values from which the above choices are to be drawn; such values will be the subject of registration. Typical sets are given in B.18.8, B.18.9 and B.18.10. The significance of multiple conditions in one FEPR is that all the conditions must be satisfied (Logical AND) for the FEPR to take effect.

A register entry for a FEPCO will define the sets for the above choices, including the ASN.1 Tags and definition of semantics, and can also define an initial set of FEPRs drawn from these sets.

20.3.6 Context Control Object (CCO)

20.3.6.1 VTE-parameters for CCO

To define a CCO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;

CO-type-identifier: the ASN.1 OBJECT IDENTIFIER value vt-b-sco-cco, see annex C, identifying the CO as being of type CCO;

CO-structure: "non-parametric";

CO-access: either of "WAVAR", "NSAC";

CO-priority: takes the value "normal";

CO-trigger: takes either valid value.

VTE-parameters specifying the characteristics of the data elements are defined in 20.3.6.2.

20.3.6.2 Definition of information field for CCO

The CCO information field consists of a sequence of six integers that are named successively as:

a) **field label z coordinate**;

b) **field label f coordinate**;

c) **field k coordinate**;

d) **entry-control index**;

e) **device-object index**;

f) **FEPR index**.

This information field is not partially updateable.

a), b) and c) together identify the field and k coordinate which indicate where data entry is to commence when a VT-user is given permission to start data entry (e.g., in S-mode by the passing of ownership of WAVAR) or where data entry has been terminated when a VT-user relinquishes permission to do data entry. See also B.18.12.

d), e) and f) are relevant only on termination and indicate the reason for termination. These items are used as follows:

- entry-control index: this is an index into the entry-control-list of the FDR addressed by the field label;
- device-object index: this is an index into the device-object-list of the entry-control addressed by the entry-control index;
- FEPR index: this is an index into the FEPR-list of the entry-control addressed by the entry-control index.

If the entry-control-list is "void", these three data elements each take value 1 (one).

If the device-object-list or FEPR-list of the entry-control has value "void", the corresponding data element takes value 1 (one).

20.3.7 Transmission Policy Control Object (TPCO)

20.3.7.1 VTE-parameters for TPCO

To define a TPCO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;
CO-type-identifier: ASN.1 OBJECT IDENTIFIER value vt-b-sco-tpco, see annex C, identifying the CO as being of type TPCO;
CO-structure: 1;
CO-access: same as CO-access for FDCO associated with the same DO, see B.18.6;
CO-priority: "normal";
CO-trigger: either valid value;
CO-category: boolean;
CO-size: 4.

20.3.7.2 Definitions of Booleans in Value of TPCO

The following booleans are defined and can be used in combination.

Boolean 1: "true" – include fields which are protected, see B.18.7.1; "false" – do not include such fields.

Boolean 2: "true" – include all fields which are not protected, see B.18.7.1; "false" – include such fields only if there has been a change to the content during the data entry activity.

Boolean 3: "true" – for any field included notify complete current content of the field; "false" – full content is not required, net effect of changes is sufficient (this is independent of value of delivery-control).

Boolean 4: "true" – update the CCO; "false" – do not update the CCO.

20.3.8 Termination Conditions Control Object (TCCO)

20.3.8.1 VTE-parameters for TCCO

To define a TCCO the CO VTE-parameters take the following values:

CO-name: any value unique within the VTE;
CO-type-identifier: a value as in table 9 to select a register entry or private type definition which defines an information content and/or update syntax for the termination event data elements, see 20.3.8.2 (d);
CO-structure: integer; 3 plus number of termination-event data elements required, see below;
CO-access: any valid value;
CO-priority: takes the value "normal";
CO-trigger: takes either valid value.

20.3.8.2 Definition of information field for TCCO

Four types of data element are defined for the TCCO; one of these types can occur a number of times if required, as defined in a VTE-profile definition.

a) **termination length data element**:

CO-element-id : takes value 1 (also used as event-id);
CO-category : takes value "integer";
CO-size : default value (maximum integer value 65535).

The content value is the number of array element updates after which termination will occur, with event-id value 1.

b) **termination timeout mantissa data element**:

CO-element-id : takes value 2 (also used as event-id for timeout);
CO-category : takes value "integer";
CO-size : default value (max integer value 65535).

The content value is used as T in the expression given in c) below.

c) **termination timeout exponent data element**:

CO-element-id : takes value 3;
CO-category : takes value "integer";
CO-size : default value (max integer value 65535).

The content value is used as E in the expression T times 10^E which yields the time in seconds, following the first update following the previous termination event, after

which termination will occur with event-id value 2. See also 23.4.

d) **termination event data element**:

This data element type is optional and can occur a number of times; this allows termination events to be categorised into groups and the groups to be individually updated and notified. The occurrence of this data element and the content of individual occurrences if appropriate will be defined in a VTE-profile or in a register entry for a TCCO.

CO-element-id : unique (to the TCCO) integer greater than 3;

CO-category : "transparent" or "character";

NOTE 1 – The choice between these value types will be made in a VTE-profile definition or in a register entry; if the choice is "character" then the VTE-parameter *CO-repertoire-assignment* becomes valid for the data element.

CO-size : as required.

If any termination event included in a data element of this type occurs it is notified with event id of value equal to the CO-element-id, see note 2.

NOTE 2 – The information in 23.4 on the use of termination conditions and notification of events in a TCO apply when termination events are specified in a TCCO.

20.3.9 Ripple mode control object RMCO

The Ripple Mode Control Object (RMCO) is used in conjunction with the ripple-capability available with the Ripple functional unit. Use of this CO requires the Structured Control Objects functional unit to be selected.

20.3.9.1 VTE-parameters for RMCO

CO-name : any value unique within the VTE;

CO-type-identifier : ASN1 OBJECT IDENTIFIER VALUE vt-b-sco-rmco, see Annex C, identifying the CO as being of type RMCO;

CO-structure : 6;

CO-access : as for the corresponding DO;

CO-priority : "normal";

CO-trigger : "not selected".

a) ripple-mode data element

CO-element-id : 1;

CO-category : "Boolean"
"true" : insert,
"false" : overwrite (the initial value);

CO-size : 1.

b) fill-mode data element

CO-element-id : 2;

CO-category : "boolean"
"true" : fill,
"false" : erase (the initial value);

CO-size : 1.

c) fill-character element

CO-element-id : 3;

CO-category : "character";

CO-repertoire-assignment : default (IRV of ISO/IEC 646);

CO-size : 1.

The initial value for fill-character is <space>.

d) ripple-limit element

CO-element-id : 4;

CO-category : "symbolic" from the set ("x", "y", "z", "p", "q", "b").

CO-size : 6.

The initial value for ripple-limit is "x".

e) logical-ripple-limit element

CO-element-id : 5;

CO-category : "symbolic" from the set ("k", "f", "z");

CO-size : 3.

The initial value for logical-ripple-limit is "k".

NOTE – The value of this element is only relevant if *field-definition--capability* is selected for the associated DO.

f) backwards-forwards element

CO-element-id : 6;

CO-category : "symbolic" from the set ("backwards", "forwards");

COsize : 2.

The initial value for backwards-forwards is "backwards".

21 Reference Information Object VTE-parameters

21.1 Availability

Availability of Reference Information Objects (RIOs) and therefore the applicability of these VTE-parameters is subject to the Reference Information Objects functional unit having been selected.

21.2 VTE-parameters for RIOs

Each RIO requires one instance of the following set of VTE-parameters to be included in the VTE. They conform to the definitions in 20.1.

CO-name: any value unique within the VTE;

CO-type-identifier: a value as in table 9 in clause 20 to select a register entry or private type definition for the initial contents of the RIO; the ASN.1 OBJECT IDENTIFIER value vt-b-sco-nullrio (see annex C) selects an empty RIO;

CO-structure: "non-parametric";

CO-access: any valid value, see clause 9 and table 1;

CO-priority: any valid value;

CO-trigger: "not-selected".

NOTE – The service provider applies no constraint on the values of *CO-access* or *CO-priority*.

22 Operations on RIOs

22.1 Availability

RIO capability and therefore the applicability of these operations is subject to the Reference Information Objects functional unit having been selected.

22.2 Identification of RIO and RIO records

22.2.1 Identification of RIO

A RIO is identified by its name *CO-name*; this is a VTE-parameter, see clause 21.

CO-name is a valid value for *VT-object-ident* parameter of VT-DATA service, see 31.1 and 22.3, when a RIO of that name is present in the current-VTE.

22.2.2 Identification of RIO-record

Each record in a RIO is identified by its name *record-id*. Record-id is not a VTE-parameter but is either known from the registered definition of a pre-defined RIO (e.g., when the RIO is called into the VTE by a profile) or is assigned by CREATE-RECORD operation, see 22.3.3. It is of value type ASN.1 PrintableString.

22.3 RIO update operations

The following RIO update operations are all subject to the access-rule applied to the RIO by *CO-access* VTE-parameter, see clause 21. They are extensions to the simple update operation defined for other control objects.

RIOs are updated using a similar principle as for other control objects, see 31.1, i.e., the *CO-name* value forms a valid value for *VT-object-ident* parameter of VT-DATA, with *VT-object-data* being comprised of the update operation and, where applicable, other arguments.

NOTE – sub-clause 19.4 defines COPY-(LOGICAL)-TO/FROM-BUFFER operations which can refer to a RIO-record as the destination or source buffer. The VT service provider does not record the alternative uses of RIO-records and the VT-users should ensure the correct usage of the content of RIO-records.

22.3.1 ERASE-RIO operation

VT-object-data takes the form

ERASE-RIO

This operation deletes all records of the RIO as in 22.3.2.

22.3.2 DELETE-RECORD operation

VT-object-data takes the form

DELETE-RECORD record-id

where record-id must be the name of an existing record in this RIO.

This operation removes the content and the record-id of the named record from the named RIO.

22.3.3 CREATE-RECORD operation

VT-object-data takes the form

CREATE-RECORD record-id object-update-information

where:

record-id can be the name of an existing record in this RIO, to be overwritten by the operation, or can be the name for a new record to be created by the operation.

object-update-information is a set of VTE object updates, i.e., as they could be included in a VT-DATA request; these objects do not need to be currently in existence; i.e., need not be part of the current-VTE. This set of updates forms the new content of the RIO record; it can be the empty set.

NOTES

1 An 'empty' update in effect erases the content of the record but does not delete it.

2 No validity checks on the record contents can be made when a record is updated since the state of the VTE at the time the record will be used is not known.

22.4 RIO reference operations

RIOs are referenced using a similar principle as for the update operations, i.e., the RIO *CO-name* forms a valid value for *VT-object-ident* service parameter of VT-DATA, with *VT-object-data* value being comprised of the reference operation and argument.

Each of these reference operations is itself subject to *CO-priority* for the RIO (see 21.2) but the effects of the updates in the referenced record are subject to the characteristics of the objects to be updated.

NOTE – see Note in 22.3.

22.4.1 EXECUTE-RECORD operation

VT-object-data takes the form

EXECUTE-RECORD record-id

where record-id must be the name of an existing record in this RIO.

This operation attempts to apply the updates which form the content of the designated record to the objects in the current-VTE, as though these updates were issued explicitly in a VT-DATA request.

This operation is not subject to the RIO access-rule. Individual updates from the record are subject to the access-rules on the objects to which they are intended to apply, with respect to the VT-user which initiated the RIO reference operation. When the operation is due to inclusion in a VT-DATA service, the relevant VT-user is the issuer of the VT-DATA request. When the operation is due to a call of a RIO via a FEPR entry reaction, see B.18.10, the relevant VT-user is the one which is executing the entry reaction.

22.4.2 CALL-RECORD operation

VT-object-data takes the form

CALL-RECORD record-id

where the argument is as in 22.4.1.

This operation has the following effects, in order:

1) The values of the display pointer, and of the logical pointer if in use, are saved;
2) An EXECUTE-RECORD operation is performed as in 22.4.1;
3) The saved pointer values are restored.

All information in 22.4.1 applies to this operation also.

23 Device object VTE-parameters

The VTE-parameters for the Basic Class VT Service device objects are defined in tables 12, 13 and 14.

23.1 Default control object VTE-parameters

The VTE-parameters **device-default-CO-access**, **device-default-CO-trigger**, **device-default-CO-priority** and **device-default-CO-initial-value** are defined in table 12. They assign values for the default control object associated with the device object, see clause 16. The default control object is always of category "boolean" with eight booleans defined. The initial value of these booleans may be set through the value of *device-default-CO-initial-value*.

23.2 Minimum Length VTE-parameters

Table 12 defines the **device-minimum-X-array-length** and **device-minimum-Y-array-length** VTE-parameters which describe the minimum acceptable imaging area for the device, i.e., this means the device must be able to handle X-arrays of device-minimum-X-array-length or longer and Y-arrays of device-minimum-Y-array-length or longer in order to support adequately the requirements of the VT-user (a device with lesser dimensions will not be suitable).

23.3 Device object VTE-parameters for Attributes

These device object VTE-parameters are provided to assign device object dependent semantics to the logical attribute values in the display object. They are defined in table 13. These override any semantics defined by the analogous VTE-parameters for the display object itself.

23.3.1 Repertoire, colour and font attributes

The device object "assignment" VTE-parameters occur as members of ordered lists and are accessed in the same way as the analogous display object VTE-parameters (see 18.2.4, 18.2.5 and 18.3). Each device object "assignment" VTE-parameter is controlled by the "capability" VTE-parameter for the display object associated with the device object.

Table 12 - General device object VTE-parameters

Parameter	Value
device-name	a character string of type ASN.1 PrintableString; see clause 16.
device-default-CO-access	any valid value for *CO-access*, see clause 9 and table 1; (default = "NSAC"); value of *CO-access* for the default control object.
device-default-CO-priority	"normal", "high", "urgent" (default = "normal"); assigns update priority to default CO subject to rules in table 11; note 2 to table 9 applies.
device-default-CO-trigger	"selected", "not-selected" (default = "not-selected"); value of *CO-trigger* for the default control object.
device-default-CO-initial-value	initial values for the eight boolean values in the default control object for the device object (default = "false" for any boolean not given an explicit initial value).
device-minimum-X-array-length	an integer value; indicates the shortest agreed (via negotiation) X-array length acceptable to both VT-users when display object data is mapped to this device; in the absence of this VTE-parameter, i.e., the default case, devices with any X-array length may be used.
device-minimum-Y-array-length	an integer value; indicates the shortest agreed (via negotiation) Y-array length acceptable to both VT-users when display object data is mapped to this device; in the absence of this VTE-parameter, i.e., the default case, devices with any Y-array length may be used.
device-control-object	multiple occurrence VTE-parameter; each instance is the *CO-name* value for a control object as defined in clause 20.
device-display-object	character string; this is identical to one of the *DO-name* VTE-parameter values, see 18.1.

NOTES

1 The *device-control-object* VTE-parameter does not include the default device control object as this control object is implicitly associated with the device object (the default control object has the same name as the device object).

2 The *device-control-object* VTE-parameter semantically links named control objects to this device object through semantics defined in a VTE-profile or by semantics agreed outside the scope of this International Standard.

3 It is recommended that the *device-control-object* VTE-parameter is used to associate instances of the control object types available with a device object and hence with a display object; see, for example, also B18.6 and B18.14.

An explicit entry in a device object "assignment" list overrides an entry in the corresponding display object "assignment" list. Thus, for example, the n-th value in the *device-repertoire-assignment* list, corresponds to the n-th value in the *repertoire-assignment* list and, if it is present, overrides the n-th value in the *repertoire-assignment* list.

A "null" value is available for use in the device object "assignment" list if it is not required to override the display object "assignment" value for a specific position in the list but subsequent positions in the list are required to contain explicit overriding values. In the absence of an explicit value in either list, the default for the VTE-parameter applies. Either list may

Table 13 - Device Object assignment VTE-parameters

Parameter	Value
device-repertoire-assignment	multiple occurrence VTE-parameter defined as for *repertoire-assignment*, (see 18.2.4).
device-font-assignment	multiple occurrence VTE-parameter defined as for *font-assignment*, (see 18.3).
device-emphasis	multiple occurrence VTE-parameter defined as for *DO-emphasis*, (see 18.2.6).
device-foreground-colour-assignment	multiple occurrence VTE-parameter defined as for *foreground-colour-assignment*, (see 18.2.5).
device-background-colour-assignment	multiple occurrence VTE-parameter defined as for *background-colour-assignment*, (see 18.2.5).

be truncated. Default values apply to entries not present in display object "assignment" lists and "null" to device object "assignment" lists.

For all these VTE-parameters, except *device-font-assignment*, a single ordered list is present in each display object and device object. In the case of font, each repertoire implied by *repertoire-capability* has a separate *font-capability* VTE-parameter in the display object and a separate list of *font-assignment* VTE-parameters in the display object and of *device-font-assignment* VTE-parameters in the device object. Thus these VTE-parameters each occur as an ordered list of ordered lists. An analogous rule of positional significance between these two lists of lists applies as in the other simpler cases, noting that in this case complete lists may be replaced by the "null" value.

23.3.2 Emphasis attributes

The VTE-parameter **device-emphasis** is provided to enable the *DO-emphasis value*, see 18.2.6, to be overridden for a particular device object. It is optional and can occur as multiple occurrences which form an ordered list; each occurrence may be referenced by its position in this list.

This International Standard does not define a significance for the value of an occurrence of this VTE-parameter. A VTE-profile definition can provide a VTE-profile argument to allow negotiation of a specific device object emphasis functionality from a more generic functionality in the VTE-profile definition. See also clause B.17.

NOTE – Such provision in a VTE-profile definition is optional.

23.4 Termination VTE-parameters

Table 14 defines the three device object termination VTE-parameters; they are optional, and only useful if a device object is associated with an object updating device (see 3.3.18). Their purpose is to specify a set of conditions under which the VT-user should cause notification of previous updates to the peer VT-user, see clause B.8.

Optionally, one or more **termination event-ids** may be defined by these parameters. The termination event-ids are used to update the Termination control object (TCO, see 20.3.1), if any, linked to the device object thus notifying the peer VT-user that an event has occurred. If a *termination event-id* is defined for a termination condition, then the occurrence of the termination condition causes the appropriate *termination event-id* to be written to the TCO (after appropriate DO and CO updates have been made). If a TCO is not defined, the peer VT-user cannot be notified of the termination condition in this way. See 20.3.1 for the standard definition of a TCO.

The notification of display object updates and of the TCO update is subject to the normal rules for the applicable type of delivery control and whether the trigger characteristic is defined for the TCO. The *CO-priority* for the TCO, if present, shall be "normal".

A termination condition does not in itself prevent further updates to the objects to which the VT-user has update access at the time. In S-mode, if a TCO is defined and has the trigger characteristic, the entry into the TCO of an event-id causes ownership of WAVAR to be reassigned and update access to be lost.

The termination parameter **device-termination-event-list** has as its value a set of zero or more pairs <event, event-id> where event-id is either a non-zero positive integer or "null". The nature of event is not defined in this International Standard. It can be a choice from a character repertoire negotiated for the display object, see note 1. It is also possible for a choice to be given in a registered VTE-profile definition (with a syntax for expressing the choice). Only in these cases is explicit negotiation of this parameter possible. It can also be defined in a VTE-profile or be private to the VT-users, see clause B.9.

The termination parameter **device-termination-length** is a single pair <length, event-id> where event-id is a non-zero positive integer or "null" and length is a positive integer denoting the number of array element updates after which this termination condition should be caused if none of the above explicit conditions has occurred first.

The termination parameter **device-termination-timeout** is a single tuple of the form

<T, E, event-id>

where T, E are integers specifying a timeout of T times 10^E seconds and event-id is a non-zero positive integer or "null". T may not be negative. If T = 0 this implies that timeout does not apply (i.e., is infinite).

Table 14 - Other Device Object VTE-parameters

Parameter	Value
device-termination-event-list	set of tuples <event, event-id>.
device-termination-length	tuple <integer, event-id>.
device-termination-timeout	tuple <integer, integer, event-id> (default = 0,0,0)

Specification of a finite timeout indicates that a termination condition is to be caused after the expiration of that time after the first update following the previous termination condition if no other termination condition occurs first.

The value of event-id does not need to be unique for each termination condition. Termination conditions may be grouped with values of event-id in any desired manner.

The access-rule on the TCO, if any, linked to a device object should be related to that for the display object with which the device object is associated in such a way that the VT-user updating the display object may also update this control object with an event-id.

Termination parameter defaults are:

a) *device-termination-event-list* absent: no explicit termination conditions are defined;

b) *device-termination-length* absent: no update number limit is defined;

c) *device-termination-timeout* absent: no timeout applies (equivalent to value T = 0).

NOTES

1 The set of events in *device-termination-event-list* may (but need not) contain some elements drawn from one or more of the character repertoires defined by *repertoire-assignment* or *device-repertoire-assignment* VTE-parameters. In such a case, the termination event is specified as <repertoire number, primary attribute value>. The primary value should be written to the display object as well as causing the termination condition. If the termination events do not correspond to "characters" from the negotiated repertoires, the events may be specified in a VTE-profile or by some other means beyond the scope of this International Standard.

2 Simple-delivery-control or quarantine-delivery-control may be used to achieve a degree of synchronisation between display object updates and TCO updates. See clause 24.

3 The use of trigger with the TCO is an alternative way of achieving a degree of synchronisation; the receipt-acknowledgement feature is then not available.

23.5 Interaction between use of TCCO or FDCO and device object VTE-parameters

If a device object is linked to a TCCO, see 20.3.8, the device termination VTE-parameters are disabled for that device object (only), but a TCO can still be used to notify occurrence of a termination event defined in the TCCO.

If a device object is linked to a FDCO which has WACI or WACA included in *CO-access*, see 20.3.3, so that the host/terminal asymmetry exists (see clause 12 and 19.5.2), then the device object termination VTE-parameters are disabled for that device object, being replaced by the features provided by the FDCO and associated FEICO and FEPCO features. If this device object is also linked to a TCCO, then the termination conditions in this TCCO are disabled for this device object (but can apply to any other device objects linked to this TCCO). If a TCO, see 20.3 and 23.4, is linked to this device object, it will not be updated by events arising from operations associated with this device object (but may be updated by events relating to other device objects linked to this TCO).

24 Delivery control, synchronisation and net-effecting

Delivery control only applies to updates to display objects and to control objects assigned "normal" update priority (display objects have implicit "normal" update priority).

Table 15 defines the VTE-parameter which controls the operation of delivery control.

Table 15 - Delivery Control VTE-parameter

Parameter	Value
type-of-delivery-control	"no-delivery-control","simple-delivery-control","quarantine-delivery-control" (default = "no-delivery-control").

For delivery control, the service provider is modelled as storing a sequence (queue) of items representing one or more updates to such objects. Each item is the content of one VT-DATA request primitive, see 31.1; when a VT-user issues a VT-DATA request, the update item is added to the tail of the stored sequence. An update item is delivered to the peer VT-user by its inclusion in a VT-DATA indication primitive; this removes the item from the head of the stored sequence.

Updates to objects which are assigned update priority other than "normal" are not subject to delivery-control and are handled on separate queues, see 24.5. Such updates will always be delivered before subsequent "normal" updates; They may overtake earlier "normal" updates if these are delayed for any reason.

24.1 No delivery-control

When the *type-of-delivery-control* VTE-parameter has the value "no-delivery-control", a VT-user entering object updates has no means of identifying significant positions in the sequence of updates. The VT-user may cause all stored update items to be delivered logically instantaneously by either

a) issuing a data transfer request primitive which addresses a control object having the *CO-trigger* VTE-parameter value "selected" (this is referred to as "trigger" characteristic), or

b) initiation of certain other services not primarily concerned with data transfer, see 24.4.

This International Standard permits the delivery of update items without occurrence of the above events. It does not limit the length of the sequence of update items which may be stored by the service provider. The maximum length of the sequence could be zero and then delivery occurs immediately.

"No-delivery-control" requires the service provider to deliver all update items in a form identical to that in which they were submitted; concatenation or segmenting of update items or "net-effecting" is not permitted.

24.2 Simple delivery-control

"Simple-delivery-control" provides the VT-user updating a display object with the additional service, VT-DELIVER, which has the following effects:

a) delivery of the stored sequence of update items occurs (thus an additional means of requiring this, relative to the "no-delivery-control" case, is available);

b) an explicit indication of the end of the sequence of update items being delivered is passed to the receiving VT-user;

c) this indication may, at the option of the VT-user initiating the delivery, also request acknowledgement of delivery by the receiving VT-user.

Apart from the above, service provider operation with "simple-delivery-control" is the same as defined for "no-delivery-control".

24.3 Quarantine delivery-control

"Quarantine-delivery-control" provides the VT-user updating a display object with the capabilities of "simple-delivery-control" with the following qualifications:

a) the service provider is not permitted to deliver update items before a deliver function, implicit or explicit, is initiated;

b) the service provider is permitted to concatenate or segment update items;

c) the service provider may "net-effect" the update items accumulated between two deliveries, see clause B.15. Net-effecting may combine updates from two or more VT-DATA request primitives provided that echo-now is not set, or is only set on the last item.

24.4 Implicit delivery

Certain confirmed services, as a side effect, have an implicit delivery effect on any stored update items. These services are VT-RELEASE, VT-SWITCH-PROFILE and VT-START-NEG. This implicit delivery action is as follows :

a) a request primitive shall force any updates issued by the initiating VT-user to be delivered to the peer VT-user before the corresponding indication primitive;

b) a response primitive indicating that the service is successful shall force any updates issued by the accepting VT-user to be delivered to the peer VT-user before the corresponding confirm primitive;

c) a response primitive indicating that the service is refused shall not force delivery of any updates but such delivery may occur at the discretion of the service provider unless "quarantine-delivery-control" is in use in which case delivery shall not occur.

NOTE – These services are described as conditionally sequenced since the response/confirm may overtake update information already issued, if the service is refused.

Implicit delivery of all updates is also caused by the passing of the WAVAR access-right either due to initiation of VT-GIVE-TOKENS service or as a result of "trigger" action, see 24.1 a).

Table 16 summarises the service provider actions with regard to delivery of updates for both A-mode and S-mode.

24.5 Update queues and priority handling

Three update priorities are defined: "normal", "high" and "urgent". Update priority is a characteristic of an object in the VTE. Display objects are implicitly of "normal" priority. A control object is assigned one of the three priorities by VTE-parameter *CO-priority*, see 20.1.

Update priority relates to the urgency with which the service provider will attempt to notify an update requested by one VT-user to the peer VT-user. "High" priority updates can be delivered when "normal" updates are held up by delivery control but can be held up by the peer VT-user or service provider back pressure. "Urgent" priority updates will be able to bypass some such blockages and will be delivered most quickly if the Urgent Data functional unit is selected, see clause 10.

The action of the service provider in performing object updates (initiated by VT-DATA, see 31.1) is modelled using a number of queues, as shown in figure 8. Two complementary instances of the mechanism are to be assumed between the pair of VT-users, each dealing with VT-DATA requests from one VT-user and giving VT-DATA indications to the peer VT-user.

NOTE – The two instances apply in both A-mode and S-mode and are inherently independent of each other. An object with access-rule "WAVAR" can only be updated by one VT-user at any particular time, since at most one VT-user owns the WAVAR access-right at any time.

The following paragraphs amplify figure 8. They apply independently to each direction of flow of updates.

24.5.1 The issue of an update request by a VT-user is subject only to the access-rule of the object.

24.5.2 A valid update request is placed at the tail of the update request queue, Q1, Q2 or Q3, corresponding to the object's update priority.

24.5.3 The three update request queues are serviced in the order "urgent", "high" and "normal", each queue being emptied completely before the next is serviced at all, i.e., after each update is taken from the head of a queue, the higher priority queues are again examined.

NOTE – The service provider will always provide the bypass priority mechanisms represented by the queues Q1 and Q4, but may restrict the number of entries in any one of these queues to one or some other small number by suspending the service interface or transfer path when this number is reached. Occupancy of Q2, Q3, Q5 or Q6 must not prevent use of the bypass mechanism.

24.5.4 A normal transfer path always exists between the two VT-users and is always used for sequenced (FIFO: first-in-first-out) transfer of update requests from "normal" or "high" queues.

24.5.5 If the Urgent Data functional unit has been selected, a special transfer path exists as well and is then always used for sequenced FIFO transfer of update requests from the "urgent"

Table 16 – Summary of actions of service provider with regard to delivery control and access-right transfer

		S–mode	A–mode
Explicit Delivery		Issue VT–DATA followed by VT–DELIVER.ind'n	Issue VT–DATA followed by VT–DELIVER.ind'n
IMPLICIT DELIVERY CONTROL	Access-right transfer	Issue VT–DATA followed by VT–GIVE–TOKENS. indication	————
	Update to Trigger CO	Issue VT–DATA followed by VT–GIVE–TOKENS. indication	Issue VT–DATA.indication
	Due to other VT-services 1. VT–RELEASE.request 2. VT–START–NEG.request 3. VT–SWITCH–PROFILE.request	Issue VT–DATA.indication followed by the indication for the requested service.	Issue VT–DATA.indication followed by the indication for the requested service.
	4. VT–RELEASE.response (accept) 5. VT–START–NEG.response (accept) 6. VT–SWITCH–PROFILE .response (accept)	Issue VT–DATA.indication followed by the confirm for the requested service	Issue VT–DATA.indication followed by the confirm for the requested service.
	7. VT–RELEASE.response (refuse) 8. VT–START–NEG.response (refuse) 9. VT–SWITCH–PROFILE .response (refuse)	Issue the confirm for the requested service (VT–DATA.indication is not forced)	Issue the confirm for the requested service (VT–DATA.indication is not forced)
NOTE – VT-DATA indication is issued only if one or more update items are present.			

queue; if it does not exist, the normal path will be used for such updates.

24.5.6 The special transfer path, if it exists, has the property that it guarantees that updates submitted to it will always be delivered before subsequent "normal" updates; They may overtake earlier "normal" updates if these are delayed for any reason.

24.5.7 The action of a stimulus to force "deliver", see 24.2, 24.3 and 24.4, causes a deliver mark to be placed in Q3 and all update request queues to be emptied to the transfer path as in 24.5.3 through 24.5.5, until the "deliver" mark is submitted to the normal path.

NOTE – An explicit stimulus to deliver by VT-DELIVER request must be distinguished in the transfer mechanism so that a VT-DELIVER indication can be generated in the correct circumstances.

24.5.8 As updates are received from the transfer path(s), they are entered into the update indication queue Q4, Q5 or Q6 corresponding to the update priority.

24.5.9 The three update indication queues are serviced in the manner described in 24.5.3, but if "quarantine-delivery-control" is in force, no updates will be taken from Q6 until a "deliver" mark is received from the transfer path, see 24.5.7 and 24.5.10. If "no-delivery-control" is in force, see 24.1, each update is issued as a VT-DATA indication to the "receiving" VT-user.

NOTE – See the note in 24.5.3.

24.5.10 A "deliver" mark causes the three update indication queues to be emptied in the order Q4, Q5, Q6, i.e., releasing any quarantining, and appropriate VT-DATA indications to be issued, with "net-effecting" allowed only if "quarantine-delivery-control" is in force.

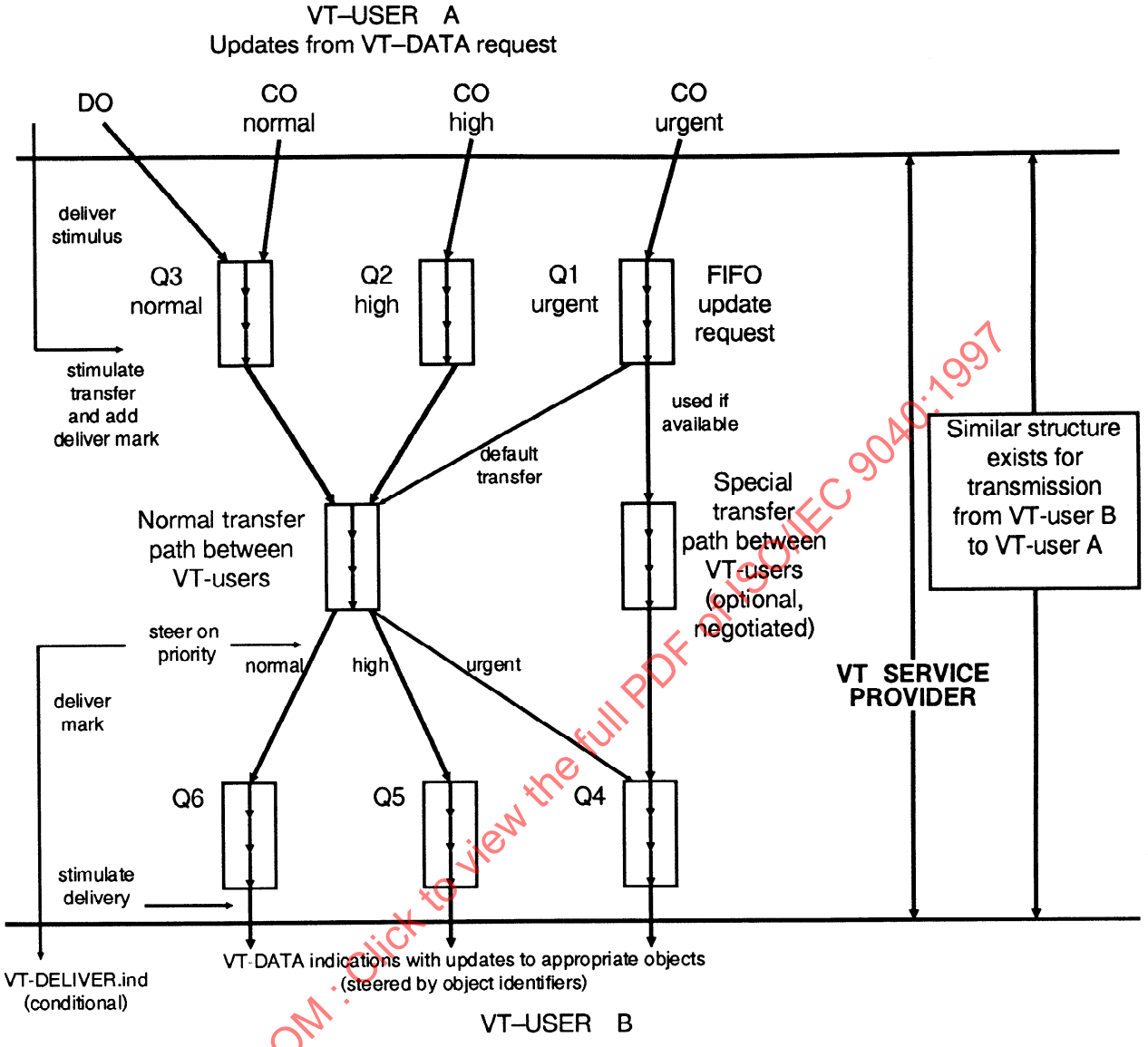


Figure 8 - Queue handling mechanism

25 Communication Model

The remaining clauses of this International Standard define the Communication Facilities of the Virtual Terminal Service.

This International Standard uses an abstract model for communication where interactions take place between the VT-users and the VT service provider. These interactions take the form of services which pass information between a VT-user and the VT service provider in VT service parameters.

NOTES

1 This model is equivalent to that in ISO/IEC 10731 except that it does not define a layer boundary or service access points.

2 This communication model is identical to that shown in figure 2 and described in clause 12.

3 VT Services are conceptual and need not be directly related to protocol elements or seen as macro calls of an access method to the service.

4 There are other equivalent sets of VT services which could provide the same service facilities.

5 Only those services relating to communication between two VT-users are defined. Services related to local conventions between the VT-user and the service provider are not within the scope of this International Standard.

26 VT Services

Table 17 lists the services of the Basic Class Virtual Terminal Service and the service facility to which each belongs. The table also indicates whether the services are

Table 17 - VT Services available with Functional Units

Functional Unit	Facility	Service	Structure	Clause
Kernel	Establishment	VT-ASSOCIATE	Cnf,Seq,Nds	28.1
	Termination	VT-RELEASE (only immediate release)	Cnf,Nsq,Nds	29.2
		VT-U-ABORT	Ncf,Nsq,Dst	29.3
		VT-P-ABORT	Prl,Nsq,Dst	29.4
	Delivery Control	VT-DELIVER	Ncf,Seq,Nds	32.1
		VT-ACK-RECEIPT	Ncf,Nsq,Nds	32.2
	Dialogue Management	VT-GIVE-TOKENS	Ncf,Seq,Nds	33.1
		VT-REQUEST-TOKENS	Ncf,Seq,Nds	33.2
	Data Transfer	VT-DATA	Ncf,Csq,Nds	31.1
Switch Profile Negotiation	Switch Profile Negotiation	VT-SWITCH-PROFILE	Cnf,Csq,Nds	30.1.1
Multiple Interaction Negotiation	Multiple Interaction Negotiation	VT-START-NEG	Cnf,Csq,Nds	30.2.1
		VT-END-NEG	Cnf,Seq,Nds	30.2.2
		VT-NEG-INVITE	Ncf,Seq,Nds	30.2.3
		VT-NEG-OFFER	Ncf,Seq,Nds	30.2.4
		VT-NEG-ACCEPT	Ncf,Seq,Nds	30.2.5
		VT-NEG-REJECT	Ncf,Seq,Nds	30.2.6
Negotiated Release	Termination	VT-RELEASE (adds negotiated release)	Cnf,Seq,Nds	29.2
Urgent Data	Data Transfer	VT-DATA (priority "urgent" COs only)	Ncf,Nsq,Nds	31.1
Break	Interrupt	VT-BREAK	Cnf,Nsq,Dst	34.1
Exceptions	Exception Reporting	VT-P-EXCEPTION	Prl,Nsq,Dst	35.1

Key for Structure abbreviations:

a) Cnf : Confirmed Service
b) Ncf : Non-confirmed Service
c) Prl : Provider-initiated Service
d) Seq : Sequenced Service
e) Nsq : Non-sequenced Service
f) Dst : Destructive Service
g) Nds : Non-destructive Service
h) Csq : Conditionally sequenced Service

a) confirmed, non-confirmed or provider initiated (see ISO/IEC 10731),

b) sequenced, conditionally sequenced or non-sequenced (see definitions in 3.3.35, 3.3.36, and 3.3.37),

c) destructive or non-destructive (see definitions in 3.3.38 and 3.3.39).

The availability of some of these VT services is subject to one or more of the optional VT functional units having been selected during VT-association establishment, see clause 10. Table 17 lists the functional units and the VT services which become available with each functional unit.

The VT services are comprised of VT service primitives. A service primitive is a distinct part of a service and is a logically instantaneous and indivisible event which may not be interrupted by another event. See ISO/IEC 10731 for definitions of the structure and other characteristics of service primitives.

The VT services defined in clauses 28 through 33 use the following notation in specifying the service parameters for each service:

M : Presence of service parameter is mandatory

O : Presence of service parameter is a user-option

C : Conditional service parameter (the text defining the service indicates whether the service parameter is present; a service parameter may be both conditional and a user-option)

A : Service parameter specified by ISO/IEC 8649 (ACSE)

= : Service parameter value is unchanged by the service provider

blank : Service parameter is absent.

NOTE – In some cases, services with a confirmed structure may not result in the issue of an indication primitive (and hence there is no response primitive). For example, when the VT-user initiates a service with service parameters in the request primitive which are not supported by the service provider, and therefore the service is completed by the service provider generating a confirm primitive (indicating failure) without involving the peer VT-user.

27 VT service sequences

27.1 Phases

The normal progress of the Basic Class Virtual Terminal service is illustrated in figure 9 in terms of the transitions between the phases of the VT service provider.

The phases of the service provider are:

a) the **Idle** phase in which no VT-association exists;

b) the **Data Handling** phase in which a VT-association is being used for the exchange of information between VT-users;

c) the **Negotiation Active** phase in which the multiple interaction negotiation (MIN) services are being used to create a new VTE;

d) the **Negotiation Quiescent** phase in which neither data handling or active MIN negotiation is in progress. It is the only valid phase when a full-VTE does not exist and MIN negotiation is not in progress.

27.2 Phase transitions

The valid transitions between the phases of the VT Service provider are shown in figure 9. The transitions are labelled T1, T2, ..., T14 and the events and conditions causing these transitions are:

T1: use of the VT-ASSOCIATE service where *VT-result* is "failure";

T2: use of the VT-ASSOCIATE service where *VT-result* is "success", i.e., a full-VTE is established;

T3: use of the VT-ASSOCIATE service where *VT-result* is "success-with-warning", i.e., a full-VTE is not established;

T4: use of VT-RELEASE service where *VT-result* is "success";

T5: use or occurrence of VT-U-ABORT or VT-P-ABORT service;

T6: use of VT-SWITCH-PROFILE service where *VT-result* is "success", i.e., the new full-VTE is established;

T7: any of the following events:

a) use of VT-RELEASE service where *VT-result* is "failure", i.e., the VT-association is retained,

b) use of VT-SWITCH-PROFILE service where *VT-result* is "failure", i.e., a new full-VTE is not established,

c) use of VT-START-NEG service where *VT-result* is "failure", i.e., a draft-VTE is not established;

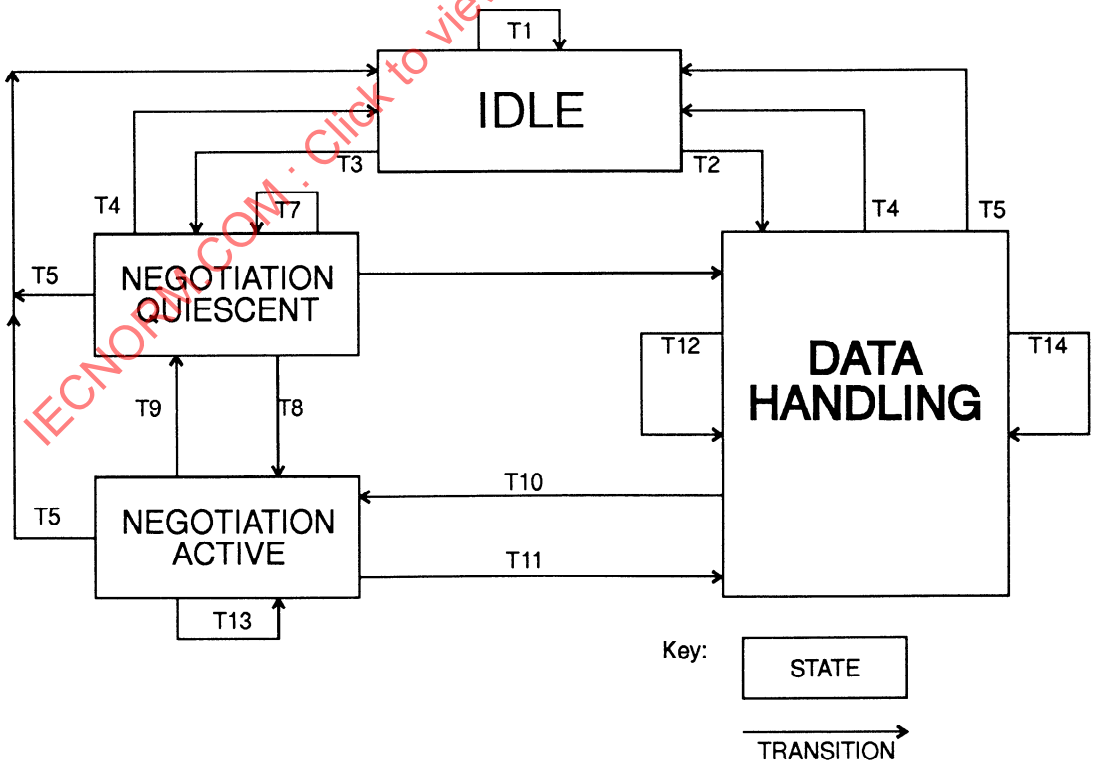


Figure 9 - Phases of the VT Service

T8: use of VT-START-NEG service where *VT-result* is "success", i.e., a draft-VTE is established (see note 1);

T9: use of VT-END-NEG service where *VT-result* is "success-with-warning", i.e., no draft-VTE now exists and a return to Data Handling phase is not agreed (or not possible) (see note 2);

T10: use of VT-START-NEG service where *VT-result* is "success", i.e., a draft-VTE is established (see note 1);

T11: use of VT-END-NEG service where *VT-result* is "success" (see note 2);

T12: use of VT-SWITCH-PROFILE (successful or unsuccessful);

T13: use of VT-END-NEG service where *VT-result* is "failure" (see note 2);

T14: use of VT-START-NEG or VT-RELEASE where *VT-result* is "failure".

Where a phase transition is caused by a confirmed service, the change of phase occurs at the initiator on receipt of the confirm and at the responder on sending the response.

Where a phase transition is caused by an unconfirmed service, the change of phase occurs on sending the request or receiving the indication as appropriate.

NOTES

1 The choices of draft-VTE for transitions T8 and T10 are defined in 30.2.1.

2 The conditions under which VT-END-NEG service results in *VT-result* values "success", "success-with-warning" and "failure" are defined in 30.2.2.

3 The precise moment when a phase change occurs is, in general, different for the two VT-users, so that the two VT-users can be transiently in different phases. The rules in 27.4 and 27.5 ensure that VT-users do not need to take account of this.

27.3 Ownership of the WAVAR access-right

In S-mode, the WAVAR access-right is owned by at most one VT-user at any one time. In A-mode, neither VT-user owns the WAVAR access-right.

The ownership of the WAVAR access-right after VT-ASSOCIATE or VT-BREAK is defined in 28.1.3 and 34.1.3. Otherwise, a VT-user becomes owner of the WAVAR access-right only on receipt of a VT-GIVE-TOKENS indication, and relinquishes ownership either by sending a VT-GIVE-TOKENS request or by updating a control object with trigger characteristic, see 31.1.4 and 33.1.4.

NOTE – While the VT-GIVE-TOKENS is in transit, neither VT-user owns the WAVAR access-right.

27.4 Availability and usage conditions of VT services

The availability of the VT services is dependent on the phase that is currently active as shown in table 18.

The following general rules also apply to the usage of the VT services:

a) VT-U-ABORT and VT-P-ABORT are not subject to any restrictions and may be initiated at any time;

b) in S-mode, a confirmed service, except VT-BREAK, and VT-DELIVER may be initiated by a VT-user only if the WAVAR access-right is owned by that VT-user;

c) no service, except VT-BREAK or VT-DATA updating "high" or "urgent" priority control objects, may be initiated by a VT-user if that VT-user has initiated a VT-DELIVER request

Table 18 - Availability of VT Services according to Phase

SERVICE	IDLE	DATA HANDLING	NEGOTIATION ACTIVE	NEGOTIATION QUIESCENT
VT-ASSOCIATE	AVAILABLE	–	–	–
VT-RELEASE	–	AVAILABLE	–	AVAILABLE
VT-U-ABORT	–	AVAILABLE	AVAILABLE	AVAILABLE
VT-P-ABORT	–	AVAILABLE	AVAILABLE	AVAILABLE
VT-SWITCH-PROFILE	–	AVAILABLE	–	AVAILABLE
VT-START-NEG	–	AVAILABLE	–	AVAILABLE
VT-END-NEG	–	–	AVAILABLE	–
VT-NEG-INVITE	–	–	AVAILABLE	–
VT-NEG-OFFER	–	–	AVAILABLE	–
VT-NEG-ACCEPT	–	–	AVAILABLE	–
VT-NEG-REJECT	–	–	AVAILABLE	–
VT-DATA	–	AVAILABLE	–	–
VT-DELIVER	–	AVAILABLE	–	–
VT-ACK-RECEIPT	–	AVAILABLE	–	–
VT-GIVE-TOKENS	–	AVAILABLE	AVAILABLE	AVAILABLE
VT-REQUEST-TOKENS	–	AVAILABLE	AVAILABLE	AVAILABLE
VT-BREAK	–	AVAILABLE	–	–
VT-P-EXCEPTION	–	AVAILABLE	–	–

primitive in which *VT-ack-request* service parameter takes value "acknowledgement" and a VT-ACK-RECEIPT indication primitive has not been received.

d) in the event of collision of VT-SWITCH-PROFILE request, VT-START-NEG request or VT-RELEASE request with VT-BREAK, the VT-BREAK wins and is performed, see 34.1. Unlike other collisions (see 27.5), a confirm primitive is not generated for the destroyed request.

e) in the event of a collision of VT-SWITCH-PROFILE.request, VT-START-NEG.request or VT-RELEASE.request with VT-P-EXCEPTION, the VT-P-EXCEPTION wins and is performed, see 35.1. Unlike other collisions (see 27.5), a confirm primitive is not generated for the destroyed request.

In S-mode, a confirmed service does not cause ownership of WAVAR to be reassigned.

27.5 Service collisions in A-mode

In A-mode, both VT-users are permitted, subject to the rules in 27.4, to issue request primitives for the confirmed services VT-RELEASE, VT-SWITCH-PROFILE and VT-START-NEG at any time. The service provider will resolve collisions of such requests in favour of one of the VT-users. The VT-user which has lost the collision receives the appropriate confirm primitive with *VT-result* value "failure" and *VT-provider-failure-reason* value "collision detected", followed by the indication primitive for the winning service.

The service provider also detects a collision when one of these service request primitives collides with a VT-DELIVER request for which the *VT-ack-request* service parameter takes value "acknowledgment". The service provider attempts to resolve this collision in favour of the initiator of the VT-DELIVER request. The VT-user which has lost the collision receives the appropriate confirm primitive with *VT-result* value "failure" and *VT-provider-failure-reason* value "collision detected". This primitive will be issued to the VT-user before the indication primitive for the VT-DELIVER (which won the collision) is issued to this VT-user. The VT-user receiving such a confirm primitive may not initiate any further primitives (other than VT-U-ABORT) until the indication for the colliding VT-DELIVER request has been received.

However, for a VT-DELIVER request where the VT functional unit Negotiated Release has not been selected, the VT-RELEASE will proceed and thus the VT-user which issued the VT-DELIVER will receive the VT-RELEASE indication instead of the expected VT-ACK-RECEIPT indication. The initiator of the VT-RELEASE will receive the VT-DELIVER indication but cannot respond to the acknowledgement request.

No implicit delivery occurs as a result of a service request rejected due to a collision.

Refer to 29.3.4 and 29.4.4 for the cases of collision with VT-U-ABORT and VT-P-ABORT.

NOTES

1 Collisions of these confirmed services with VT-DELIVER do not occur in S-mode since the rules in 27.4 use ownership of WAVAR as a condition for the initiation of any of these services.

2 A VT-DELIVER which does not have VT-ack-request with value "acknowledgement" does not cause collision. Both services (i.e., the VT-DELIVER and the confirmed service) are allowed to proceed independently with their normal effects.

3 A collision between VT-BREAK and any other service primitive, except VT-P-ABORT, VT-U-ABORT and VT-P-EXCEPTION is resolved in favour of the VT-user issuing the VT-BREAK.

4 The VT-user which issued a VT-DELIVER request which won a collision receives no notification that a collision occurred.

5 A collision between VT-P-EXCEPTION and any other service primitive, except VT-P-ABORT, VT-U-ABORT and VT-BREAK is resolved in favour of the VT-P-EXCEPTION.

28 Establishment facility

This clause defines the VT service which supports the Establishment facility, see 7.1.

28.1 VT-ASSOCIATE service

28.1.1 Purpose

To establish a VT-Association.

28.1.2 Structure

See table 17.

28.1.3 Service Parameters

Table 19 specifies the service parameters for the VT-ASSOCIATE service together with an indication of when each parameter is required, see clause 26.

28.1.3.1 Called Application Entity Title is the application entity title of the application entity with which a VT-association is to be established.

Table 19 - VT-ASSOCIATE service parameters

Parameter Name	Req.	Ind.	Rsp.	Cfm.
Called Application Entity Title	A	A=		
Calling Application Entity Title	A	A=		
Responding Application Entity Title			A	A=
VT-class	M	M=		
VT-functional-units	O	O	O	O
VT-mode	M	M=		
VT-WAVAR-owner	C	C=	C	C=
VTE-profile-name	O	O=		
VTE-profile-arg-offer-list	C	C		
VTE-profile-arg-value-list			C	C=
VT-result			M	M
VT-user-failure-reason			C	C=
VT-provider-failure-reason				C

28.1.3.2 Calling Application Entity Title is the application entity title of the application entity initiating this VT-association.

28.1.3.3 Responding Application Entity Title is the application entity title of the application entity responding to a VT-association request.

28.1.3.4 VT-class specifies the virtual terminal service class to be used on the VT-association; for this International Standard, it is always set to the value "BASIC".

28.1.3.5 VT-functional-units is an optional service parameter which allows the VT-users to negotiate which functional units of the VT service will be used, see clause 10. Its value is a list taken from the symbolic values contained in the following list:

a) Switch Profile Negotiation
b) Switch Profile and MIN negotiation
c) Negotiated Release
d) Urgent Data
e) Break
f) Enhanced Access-rules
g) Structured Control Objects
h) Blocks
i) Fields
j) Reference Information Objects
k) Ripple
l) Exceptions
m) Context Retention

Each VT-user specifies, in request or response primitive as appropriate, the set of functional units it has available and is willing to use. The service provider may reduce these sets and if so will remove the same functional units from each set before issuing the indication or confirm primitive. The set of functional units available for the VT-association is that which is common to the indication and confirm primitives.

28.1.3.6 VT-mode is a mandatory service parameter which enables the initiating VT-user to select the initial mode of operation and whether mode switching will be allowed during the lifetime of the VT-association. There is no return value in the response or confirmation primitives; if the peer VT-user cannot accept the value, the association must be refused. It takes a value from the set ("S-mode", "A-mode", "either-S", "either-A").

Values "S-mode" and "either-S" initiate a S-mode VT-association; values "A-mode" and "either-A" initiate an A-mode VT-association, see clause 8. Values "S-mode" and "A-mode" do not allow mode switching; values "either-S" and "either-A" do allow mode switching. When one of "either-S" or "either-A" is in use, a VT-user may switch modes for a VT-association by using the negotiation facilities to negotiate a new VTE based on a VTE-profile which specifies another mode. This service parameter implicitly determines the access-rules which are initially available for the VT-association (see clauses 8 and 9).

28.1.3.7 VT-WAVAR-owner is an optional and conditional service parameter which enables the VT-users to negotiate the initial ownership of the WAVAR access-right. It is only applicable if the value of service parameter VT-mode is "S-mode" or "either-S". The value in a request/indication is one of the values:

a) "initiator-side";
b) "acceptor-side";
c) "acceptor-chooses".

When the initiating VT-user designates the initial owner of the WAVAR access-right, i.e., a) or b), the response/confirm primitives contain the same values as the request/indication primitives; when the initiating VT-user designates c), the accepting VT-user replies with his choice from a) and b). If this service parameter is omitted from the request, it is treated as if "acceptor chooses" had been specified. The parameter may only be omitted from the response if it was omitted from the request; if omitted, it is treated as if "initiator-side" were specified.

28.1.3.8 VTE-profile-name is an optional service parameter. If present, it is the name of a VTE-profile which forms the basis for the initial VTE for the VT-association. When a named VTE-profile is parameterised, values for the VTE-profile arguments must be agreed by the VT-users before a full-VTE may be formed. If absent, the service provider selects the default VTE-profile appropriate to the mode of operation, see annex A.

If the value of *VT-mode* is "A-mode" or "either-A", any value of the *VT-profile-name* parameter which specifies an S-mode profile is invalid. If the value of *VT-mode* is "S-mode" or "either-S", any value of the *VT-profile-name* parameter which specifies an A-mode profile is invalid.

Note – This rule applies to VT-ASSOCIATE only, not to VT-SWITCH-PROFILE nor to VT-START-NEG which reference this subclause.

28.1.3.9 VTE-profile-arg-offer-list may only be used if the named VTE-profile is a parameterised VTE-profile. This service parameter is a list of items where each item in the list must be one of the following:

– a single VTE-profile-argument value;
– a list of VTE-profile-argument values;
– a range of VTE-profile-argument values.

Each item is identified by its VTE-parameter identifier or its special-profile-argument identifier as appropriate, see the *VTE-profile-arg-value-list* service parameter. The items in this list supply the "offer" values for the VTE-profile arguments in a parameterised VTE-profile and may not include other VTE-parameters.

NOTE – Where a VTE-profile so permits, a particular VTE-profile argument can occur in a number of different items. These represent distinct offers for a multiple capability as defined in the VTE-profile definition; each can itself be a multiple offer.

It is permissible to omit this service parameter, in which case the peer VT-user may choose any valid values for the VTE-profile arguments, i.e., as defined in the VTE-profile definition. Similarly, if the item for any VTE-profile argument is absent, the choice is left open.

28.1.3.10 VTE-profile-arg-value-list is a list of items where each item is a single value for a profile argument identified by its VTE-parameter identifier or its special-profile-argument identifier. Where the value of *VT-result* is "success", this is a mandatory parameter for the reasons stated in the note. When the value of *VT-result* is "success-with-warning", this is an optional parameter as missing argument values may be supplied through negotiation (defaults are not assumed for the reasons stated in the Note). When the value of *VT-result* is "failure", this parameter is not present as the values it contains are not needed.

The items in the list supply the "accept" values for the VTE-profile arguments and may not include other VTE-parameters nor contain values outside the range of freedom for the corresponding item in the *VT-profile-arg-offer-list*.

NOTE – The service provider may itself reject a unique value in the *VTE-profile-arg-offer-list* or reduce a list or range, but does not alter the values in *VTE-profile-arg-value-list*. Since the VT-user which receives the indication primitive is not able to tell whether or not a unique value is due to the service provider reducing a more open offer to the unique value, it is necessary for the value list to be always returned in case the request issuer did make a multiple offer and thus needs to know the choice.

Items in this list are of the form <VTE-param-ident, VTE-parameter-value> where VTE-param-ident is the name for a VTE-parameter in the directed graph and VTE-parameter-value is a valid value for the VTE-parameter. In the case where an item is a special-profile-argument (i.e., is not from the directed graph), it takes the form <special-profile-arg-ident, special-profile-arg-value> where special-profile-arg-ident has the symbolic form "Pp-n" where n is an integer defined in the VTE-profile definition and special-profile-arg-value is a valid value as defined in the VTE-profile definition. List items in the *VT-profile-arg-offer-list* and *VT-profile-arg-value-list* service parameters for special-profile-arguments are always one of the ASN.1 primitive types BOOLEAN, INTEGER, PrintableString or OBJECT IDENTIFIER.

28.1.3.11 VT-result indicates the result of the service; it takes one of the values "success", "success-with-warning" or "failure". If the value is "success", neither *VT-user-failure-reason* or *VT-provider-failure-reason* is present.

28.1.3.12 VT-user-failure-reason is optional and may be provided if *VT-result* is not "success", due to the VT-user rejecting the VT-ASSOCIATE. It can contain either (or both) a symbolic value as defined for VT-provider-failure-reason, except "VT-ASSOCIATE-data-too-long", or an ASN.1 PrintableString.

28.1.3.13 VT-provider-failure-reason contains the provider supplied rejection reason if *VT-result* is not "success", due to the service provider rejecting the VT-ASSOCIATE; values are symbolic:

- "VTE-incomplete";
- "VTE-parameter-not-supported";
- "VTE-parameter-combination-not-supported";
- "VTE-profile-not-supported";
- "VT-ASSOCIATE-data-too-long";
- "mode-of-operation-not-supported".

28.1.4 Usage

The rules of 27.4 apply to this service.

This service may be initiated by a VT-user at any time; subsequently, no further service requests are accepted from the initiating VT-user until one of the following occurs:

a) the VT-association is accepted by the peer VT-user and is thus established (*VT-result* value is "success" or "success-with-warning");

b) the VT-association is refused by the peer VT-user returning *VT-result* value "failure";

c) the VT-association is refused by the service provider returning *VT-result* value "failure";

d) the VT-association establishment is terminated by either VT-user initiating the VT-U-ABORT service;

e) VT-association establishment is terminated by the service provider initiating the VT-P-ABORT service.

NOTE – While it is not enforced, the recommended method for an accepting VT-user to refuse to establish a VT-association is to set *VT-result* to "failure" in a VT-ASSOCIATE response. (Though not recommended, a VT-U-ABORT may also be used.)

28.1.5 Effects:

28.1.5.1 The service parameter values given in a response primitive are delivered unchanged to the peer VT-user in a confirm primitive and are deemed to apply to the VT-association if this has been accepted.

28.1.5.2 If *VT-result* has the value "success" in VT-ASSOCIATE confirm, the data handling phase is entered. If the request contained any open choices, lists or ranges in the *VTE-profile-arg-offer-list*, a response with the value "success" is required to contain specific values for these chosen from the offered lists or ranges. The VT-context-value is set to its initial value, see 13.1 and 20.2.4.

28.1.5.3 If the accepting VT-user is unable to find an acceptable choice from any of the lists or ranges of the *VT-profile-arg-offer-list* or cannot accept a single offered value, a response "success-with-warning" may be used, provided at least one of the negotiation VT functional units has been selected. This indicates a transfer to the Negotiation Quiescent phase.

28.1.5.4 If *VT-result* has value "failure" in the VT-ASSOCIATE confirm, all effects of the service are cancelled and there is no change of phase. The initiation of VT-U-ABORT by either VT-user, or VT-P-ABORT by the service provider, has a similar effect.

28.1.5.5 The value of *VT-mode* service parameter determines the initial mode of operation and whether this can be changed during the VT-association.

29 Termination facility

This clause defines the VT services which support the termination facility, see 7.2.

29.1 Services

Termination is achieved using one of the three termination services:

a) VT-RELEASE service, which enables a user-requested orderly termination of the VT-association with no loss of data; this is refusable if the appropriate functional unit is selected (see 29.2);

b) VT-U-ABORT service, which effects a user-requested immediate termination of the VT-association with possible loss of data (see 29.3);

c) VT-P-ABORT service, which notifies the users of an immediate termination of the VT-association by the service provider with possible loss of data (see 29.4).

29.2 VT-RELEASE service

29.2.1 Purpose

To terminate the VT-association. The VT-Association may be terminated in an agreed and orderly manner if the Negotiated Release functional unit is selected. If the Negotiation Release functional unit is not selected, the release is unconditional.

29.2.2 Structure

See table 17.

29.2.3 Service parameters

Table 20 specifies the service parameters for the VT-RELEASE service together with an indication of when each parameter is required, see clause 26.

Table 20 - VT-RELEASE service parameters

Parameter Name	Req.	Ind.	Rsp.	Cfm.
VT-result			M	M=
VT-user-failure-reason			C	C=
VT-provider-failure-reason				C

29.2.3.1 VT-result takes one of two values, "success" and "failure". If *VT-result* has value "success", neither *VT-user-failure-reason* nor *VT-provider-failure-reason* take a value.

29.2.3.2 VT-user-failure-reason may optionally contain the VT-user supplied rejection reason as an ASN.1 PrintableString if *VT-result* is "failure", due to the VT-user rejecting the VT-RELEASE.

29.2.3.3 VT-provider-failure-reason contains the provider supplied rejection reason if *VT-result* is "failure", due to the service provider rejecting the VT-RELEASE; value is symbolic:

– "collision-detected" (A-mode only).

29.2.4 Usage and effects

The rules of 27.4 and 27.5 apply to this service.

If *VT-result* is "success", the VT-association is terminated and the VTE and VT-context-value become undefined. If *VT-result* is "failure" due to the VT-user receiving the indication primitive refusing to accept the Release, the VT Service is restored to the VT-context-value existing before the service was initiated; the *VT-user-failure-reason* service parameter may optionally be used to convey a reason for the refusal. Refusal by the VT-user is possible only if Negotiated Release functional unit is selected.

Clause 24 defines the implicit delivery caused by this service.

NOTE – If the Negotiated Release functional unit is not selected the release request cannot be refused but the accepting VT-user determines when the release actually occurs, i.e., this is not until the response primitive is issued.

29.3 VT-U-ABORT service

29.3.1 Purpose

To terminate the VT-association with immediate effect.

29.3.2 Structure

See table 17.

29.3.3 Service parameter

Table 21 specifies the service parameter for the VT-U-ABORT service together with an indication of when the parameter is required, see clause 26.

Table 21 - VT-U-ABORT service parameter

Parameter Name	Req.	Ind.		
VT-user-failure-reason	O	O=		

VT-user-failure-reason may optionally contain the VT-user supplied rejection reason as a non-null ASN.1 PrintableString.

29.3.4 Usage and effects

A VT-user may issue a VT-U-ABORT request primitive at any time. Effects are not necessarily sequenced with regard to other services; the request may be given priority by the service provider and therefore be destructive of other VT services. The VT-association is broken and the VTE and VT-context-value become undefined.

Use of this service takes precedence over and cancels any other service with which it collides except VT-P-ABORT and VT-U-ABORT. The VT-U-ABORT indication is delivered unless that VT-user has issued a VT-RELEASE response with *VT-result* "success" or VT-ASSOCIATE response with *VT-result* "failure".

In case of collision of two VT-U-ABORT requests, neither VT-U-ABORT indication primitive is delivered, but the service termination does occur.

If a VT-U-ABORT request collides with an occurrence of VT-P-ABORT, the VT-U-ABORT indication is superseded, see 29.4.4.

NOTE – Due to these potential collisions, delivery of the *VT-user-failure-reason* service parameter content is not guaranteed and non-delivery is not notified to the issuing VT-user.

29.4 VT-P-ABORT service

29.4.1 Purpose

To notify the VT-users that the VT-association has been terminated due to an irrecoverable VT service provider or lower layer service exception condition.

29.4.2 Structure

See table 17.

29.4.3 Service parameter

Table 22 specifies the service parameter for the VT-P-ABORT service together with an indication of when the parameter is required, see clause 26.

Table 22 - VT-P-ABORT service parameter

Parameter Name	Ind.	Ind.		
VT-reason	C	C		

VT-reason contains the reason for the abort if one is available from the entity generating the abort; values are symbolic:

- "VT-protocol-error",
- "local-error".

29.4.4 Usage and effects

This service is available at any time. Effects are destructive over any other service which may be in process of execution. A VT-P-ABORT indication is not delivered to a VT-user which has issued a VT-U-ABORT request, a VT-RELEASE response with *VT-result* "success" or VT-ASSOCIATE response with *VT-result "failure"*. The VT-association is broken and the VTE and VT-context-value becomes undefined.

30 Negotiation facilities

This clause defines the VT services which support the negotiation facilities, see 7.3 and 10.1.

30.1 Switch Profile negotiation

30.1.1 VT-SWITCH-PROFILE service

30.1.1.1 Purpose

To negotiate a switch to a full-VTE constructed from a named VTE-profile.

30.1.1.2 Structure

See table 17.

30.1.1.3 Service parameters

Table 23 specifies the service parameters for the VT-SWITCH-PROFILE service together with an indication of when each parameter is required, see clause 26.

Table 23 - VT-SWITCH-PROFILE service parameters

Parameter Name	Req.	Ind.	Rsp.	Cfm.
VTE-profile-name	O	O=		
VTE-profile-arg-offer-list	C	C		
VTE-profile-arg-value-list			C	C=
VT-result			M	M=
VT-user-failure-reason			C	C=
VT-provider-failure-reason				C
VT-object-retention-list	O	C	C	C=

30.1.1.3.1 The definition and use of service parameters **VTE-profile-name**, **VTE-profile-arg-offer-list**, and **VTE-profile-arg-value-list** is identical to that defined for VT-ASSOCIATE in 28.1.3. The service provider may reduce the values or ranges contained in the request primitive before issuing the indication primitive.

30.1.1.3.2 VT-result takes one of the values "success" or "failure". If VT-result has a value "success", neither *VT-user-failure-reason* or *VT-provider-failure-reason* are present.

30.1.1.3.3 VT-user-failure-reason is optional and may be provided if *VT-result* is "failure", due to the VT-user rejecting the VT-SWITCH-PROFILE. It can contain either (or both) a symbolic value as defined for *VT-provider-failure-reason*, except "collision detected", or an ASN.1 PrintableString.

30.1.1.3.4 VT-provider-failure-reason contains the provider supplied rejection reason if *VT-result* is "failure", due to the service provider rejecting the VT-SWITCH-PROFILE; values are symbolic:

- "collision-detected" (A-mode only);
- "VTE-parameter-value-not-supported";
- "VTE-parameter-value-combination-not-supported";

– "VTE-profile-not-supported";

– "VTE-incomplete".

30.1.1.3.5 VT-object-retention-list may only be present if the Context Retention functional unit is selected. This parameter is then optional and if present in a request contains a list of the names of those DOs and COs whose contents are requested to be retained. This parameter is present in the indication and response if and only if it is present in the request. It is present in the response or confirmation if and only if it is present in the request and VT-result="success". The service provider may reduce this list as provided in the request depending on its capability and on the specific items being negotiated. The list may be reduced to the empty list. The peer VT-user may further reduce this list. The service provider must not change this parameter as provided in the response when delivering the confirmation.

30.1.1.4 Usage and effects

The rules of 27.4 and 27.5 apply to this service.

The VT-context-value is saved while the VT-SWITCH-PROFILE service is performed. *VT-result* value "success" indicates that the requested switch of profile has been made; the reset-context is set from the new full-VTE. The new full-VTE is installed as current-VTE with the VT-context-value set to its initial value for this full-VTE, see 13.1.1 and 20.2.4, except that objects listed in VT-object-retention-list take their values from the saved VT-context-value.

If a full-VTE, acceptable to both the service provider and the accepting VT-user, cannot be constructed from the service parameters in the request primitive, then the service must be refused with *VT-result* value "failure". All effects of the service are annulled and the VT-context-value is restored as though the request had never been issued except that the provisions in clause 24 for implicit delivery apply to this service.

The *VT-object-retention-list* parameter is used when it is desired to retain the contents of certain VT objects across an instance of the VT-SWITCH-PROFILE service. The service provider and/or the peer VT-user may decide that it is not possible to retain the contents of specific items listed in this parameter; these items shall be removed from the VT-object-retention-list parameter. For instance:

a) if the result of the use of this service is a change to the VTE-parameters associated with a particular object, then the contents of that object may not be retained;

b) some COs are part of a complex linked data structure. Where that structure is changed as a result of the use of this service, it may not be possible to retain the contents of certain COs which are part of that linked structure.

When this service is completed and the resultant mode is S-mode, the ownership of the WAVAR access right resides with the initiator of the service.

30.2 Multiple interaction negotiation

30.2.1 VT-START-NEG service

30.2.1.1 Purpose

To negotiate a transition to Negotiation Active phase and, optionally, name a VTE-profile to be used as the initial draft-VTE for MIN.

30.2.1.2 Structure

See table 17.

30.2.1.3 Service parameters

Table 24 specifies the service parameters for the VT-START-NEG service together with an indication of when each parameter is required, see clause 26.

Table 24 - VT-START-NEG service parameters

Parameter Name	Req.	Ind.	Rsp.	Cfm.
VTE-profile-name	O	O=		
VTE-profile-arg-offer-list	C	C		
VTE-profile-arg-value-list			C	C=
VT-result			M	M=
VT-user-failure-reason			C	C=
VT-provider-failure-reason				C

30.2.1.3.1 The use of service parameters **VTE-profile-name**, **VTE-profile-arg-offer-list** and **VTE-profile-arg-value-list** is identical to that defined for VT-ASSOCIATE in 28.1.3. The VTE constructed from these service parameters is used as the initial draft-VTE in MIN if the service is successful.

If the *VTE-profile-name* service parameter is not supplied, the current-VTE is used as the initial draft-VTE.

If the VT-ASSOCIATE service returned a *VT-result* "success with warning" and *VT-provider-failure-reason* "VTE-incomplete", the current-VTE may not be a full-VTE. In this case, the current-VTE is defined by the VTE-profile named in the VT-ASSOCIATE exchange together with those VTE-parameter values which were accepted from the *VTE-profile-arg-value-list*; those VTE-parameters, for which values were not agreed (i.e., had "null" values specified in the value list) during the VT-association exchange, are undefined.

30.2.1.3.2 VT-result takes one of the values "success" or "failure". If VT-result has a value "success", neither *VT-user-failure-reason* or *VT-provider-failure-reason* are present.

30.2.1.3.3 VT-user-failure-reason is optional and may be provided if *VT-result* is "failure", due to the VT-user rejecting the VT-START-NEG. It can contain either (or both) a symbolic value as defined for *VT-provider-failure-reason*, except "collision detected", or an ASN.1 PrintableString.

30.2.1.3.4 VT-provider-failure-reason contains the provider supplied rejection reason if *VT-result* is "failure", due to the

service provider rejecting the VT-START-NEG; values are symbolic:

- "collision detected" (A-mode only);
- "VTE-profile not supported".

NOTES

1 The VTE resulting from the service parameters in this service does not need to be complete or acceptable to both VT-users since the Negotiation Active phase enables further changes to be agreed before the VTE is used as a full-VTE.

2 Special profile arguments need to be negotiated with the VT-START-NEG service when entering the Negotiation Active phase as these arguments are not valid with other MIN services (see 30.3).

3 If *VTE-profile-name* is not present and the current-VTE is used as the initial draft-VTE, service parameters *VTE-profile-arg-offer-list* and *VTE-profile-arg-value-list* are not used (see 28.1.3).

30.2.1.4 Usage and effects

The rules of 27.4 and 27.5 apply to this service.

VT-result value "success" indicates that the requested change to the Negotiation Active phase has been made. For a *VT-result* with value "failure", the only effect of the service is implicit delivery as defined in clause 24.

The VT-context-value is saved during the execution of the VT-START-NEG service and the subsequent services up to and including the VT-END-NEG so that this VT-context-value may be restored if a switch to use the draft-VTE resulting from the negotiation as the new full-VTE is not made.

30.2.2 VT-END-NEG service

30.2.2.1 Purpose

To effect a synchronised termination of MIN and a transition to the Data Handling phase.

NOTE – In some circumstances, termination of MIN may not occur or transition may be to Negotiation Quiescent phase.

30.2.2.2 Structure

See table 17.

30.2.2.3 Service parameters

Table 25 specifies the service parameters for the VT-END-NEG service together with an indication of when each parameter is required, see clause 26.

30.2.2.3.1 VT-vte-choice indicates which of the available VTEs may be chosen as current-VTE. It takes one of the values "draft", "current" or "either". This parameter must be present on the response if it has value "either" on the request and if *VT-result* does not have value "failure"; it must not be present in other cases. On the response, the value must not be "either".

30.2.2.3.2 VT-failure-allowed is optional and takes one of the values "yes" or "no". "No" indicates the initiator is forcing negotiation to end, i.e., return is to either the Data Handling phase or to the Quiescent phase. "Yes" indicates the responder may set *VT-result* to "failure" in order to remain in the Negotiation Active phase. If not present, a value "no" is assumed for this parameter. This parameter does not influence whether or not the service provider may set *VT-result* to "failure".

Table 25 - VT-END-NEG service parameters

Parameter Name	Req.	Ind.	Rsp.	Cfm.
VT-vte-choice	M	M	C	C=
VT-failure-allowed	O	O=		
VT-result			M	M=
VT-user-failure-reason			C	C=
VT-provider-failure-reason				C
VT-object-retention-list	O	C	C	C=

30.2.2.3.3 VT-result takes one of the values "success", "success-with-warning", or "failure". If VT-result has a value "success", neither *VT-user-failure-reason* or *VT-provider-failure-reason* take a value.

30.2.2.3.4 VT-user-failure-reason is optional and may be provided if *VT-result* is not "success", due to the VT-user rejecting the VT-END-NEG. It can contain either (or both) a symbolic value as defined for *VT-provider-failure-reason* or an ASN.1 PrintableString.

30.2.2.3.5 VT-provider-failure-reason contains the provider supplied rejection reason if *VT-result* is not "success", due to the service provider rejecting the VT-END-NEG; value is symbolic:

- "VTE-incomplete".

30.2.2.3.6 VT-object-retention-list is defined as for the corresponding parameter on VT-SWITCH-PROFILE, see 30.1.1.

30.2.2.4 Usage and effects

The rules of 27.4 and 30.3 apply to this service.

If *VT-vte-choice* has value "draft" in the request and the draft-VTE is incomplete, the service provider will not generate an indication, but will return a confirm with *VT-result* = "failure" and *VT-provider-failure-reason* = "VTE-incomplete".

If *VT-vte-choice* has value "either" in the request and the draft-VTE is incomplete, then the service provider will set *VT-vte-choice* on the indication to "current".

The service provider will otherwise deliver the indication with parameters equal to those on the request; the confirm will always be identical to the response.

The resulting phase is determined by the value of *VT-result* on the response and confirm:

a) If *VT-result* = "success", the Data Handling phase is entered;

b) If *VT-result* = "failure", the service remains in the Negotiation Active phase and the draft-VTE is retained;

c) If *VT-result* = "success-with-warning", the draft-VTE is discarded and the Negotiation Quiescent phase is entered. This case only arises if the chosen VTE is the current-VTE and the current-VTE is incomplete.

In case a), the new current-VTE is determined by the value of *VT-vte-choice*. If the value on the indication or confirmation is "draft", then the reset-context is set from the draft-VTE. The draft-VTE is installed as current-VTE with the VT-context-value set to its initial value for this full-VTE, see 13.1.1 and 20.2.4, except that objects listed in *VT-object-retention-list* take their values from the saved VT-context-value. If the value on the indication or confirmation is "current", then the draft-VTE is discarded and the previous VTE remains current.

NOTE – If *VT-failure-allowed* has value "no", the only allowed values for *VT-result* are "success" and "success with warning".

The *VT-object-retention-list* parameter is used when it is desired to retain the contents of certain VT-objects across an instance of multiple interaction negotiation when the result is the adoption of a new full-VTE. The service provider and/or the peer VT-user may decide that it is not possible to retain the contents of specific items listed in this parameter; these items shall be removed from the *VT-object-retention-list* parameter. For instance:

- if the result of the use of this service is a change to the VTE-parameters associated with a particular object, then the contents of that object may not be retained;
- some COs are part of a complex linked data structure. Where that structure is changed as a result of the use of this service, it may not be possible to retain the contents of certain COs which are part of that linked structure.

When this service is completed and the resultant mode is S-mode, the ownership of the WAVAR access right resides with the initiator of the service.

30.2.3 VT-NEG-INVITE service

30.2.3.1 Purpose

To invite a peer VT-user to propose values for one or more VTE-parameters.

30.2.3.2 Structure

See table 17.

30.2.3.3 Service parameter

Table 26 specifies the service parameter for the VT-NEG-INVITE service together with an indication of when the parameter is required, see clause 26.

VT-param-ident-list is a list of VTE-parameter identifiers (names). The service provider delivers an indication to the peer VT-user containing the same VTE-parameter identifiers as were in the request.

Table 26 - VT-NEG-INVITE service parameter

Parameter Name	Req.	Ind.		
VT-param-ident-list	M	M=		

30.2.3.4 Usage and effects

The rules of 27.4 and 30.3 apply to this service.

30.2.4 VT-NEG-OFFER service

30.2.4.1 Purpose

To propose a set of acceptable (to the initiating VT-user) VTE-parameter values and/or value ranges to the peer VT-user.

30.2.4.2 Structure

See table 17.

30.2.4.3 Service parameter

Table 27 specifies the service parameter for the VT-NEG-OFFER service together with an indication of when the parameter is required, see clause 26.

Table 27 - VT-NEG-OFFER service parameter

Parameter Name	Req.	Ind.		
VT-param-offer-list	M	M		

VT-param-offer-list is a list of items where each item of the list is one of the following:

a) a single VTE-parameter value;

b) a range of VTE-parameter values;

c) a list whose elements are each either a single VTE-parameter value or a range of VTE-parameter values.

Each item is identified by its VTE-parameter identifier (name). The items in this list supply the proposed values for negotiation. The service provider delivers an indication to the peer VT-user in which the *VT-param-offer-list* list or range of VTE-parameter values may have been reduced to take into account its own capabilities. If the service provider has to remove all the offered values for any VTE-parameter in a *VT-param-offer-list* list or range, it inserts for each such VTE-parameter the symbolic value "reduced-to-null".

30.2.4.4 Usage and effects

The rules of 27.4 and 30.3 apply to this service.

When a VTE-parameter with value "reduced-to-null" is received in a VT-NEG-OFFER indication, the VT-user should either propose a counter offer for that VTE-parameter or reject it in order to complete the MIN sequence for that VTE-parameter (see 30.2.6 and 30.3).

30.2.5 VT-NEG-ACCEPT service

30.2.5.1 Purpose

To select values for one or more VTE-parameters from those proposed in VT-NEG-OFFER indication primitives.

30.2.5.2 Structure

See table 17.

30.2.5.3 Service parameter

Table 28 specifies the service parameter for the VT-NEG-ACCEPT service together with an indication of when the parameter is required, see clause 26.

Table 28 - VT-NEG-ACCEPT service parameter

Parameter Name	Req.	Ind.		
VT-param-value-list	M	M=		

VT-param-value-list is a list of items each consisting of a VTE-parameter identifier and a value for this VTE-parameter. The items in this list select values for individual VTE-parameters and are the set or a subset of the VTE-parameters proposed in previous VT-NEG-OFFER(s) with VTE-parameter values being in all cases within any value list or range(s) offered. A VTE-parameter may not be included in this service if it had the value "reduced-to-null" in the VT-NEG-OFFER indication. The service provider delivers an indication to the peer VT-user containing the same VTE-parameter values that were in the request.

30.2.5.4 Usage and effects

The rules of 27.4 and 30.3 apply to this service.

30.2.6 VT-NEG-REJECT service

30.2.6.1 Purpose

To reject values for one or more VTE-parameters from those proposed in VT-NEG-OFFER indication primitives.

30.2.6.2 Structure

See table 17.

30.2.6.3 Service parameter

Table 29 specifies the service parameter for the VT-NEG-REJECT service together with an indication of when the parameter is required, see clause 26.

Table 29 - VT-NEG-REJECT service parameter

Parameter Name	Req.	Ind.		
VT-param-ident-list	M	M=		

VT-param-ident-list is a list whose items are VTE-parameter identifiers (names). The items in the list are the set or a subset of the VTE-parameter identifiers proposed in previous VT-NEG-OFFER(s). The VT-user must reject any VTE-parameter where the value for that VTE-parameter in the VT-NEG-OFFER indication was "reduced-to-null" unless a counter offer is proposed for that VTE-parameter (see 30.2.4). The service provider delivers an indication to the peer VT-user containing the same VTE-parameter identifiers as were in the request.

30.2.6.4 Usage and effects

The rules of 27.4 and 30.3 apply to this service.

30.3 Sequence control for multiple interaction negotiation

The VT services valid for use in the Negotiation Active phase are defined in 30.2.3 to 30.2.6. A sequence of these services may be used to negotiate a new value for a VTE-parameter while in the Negotiation Active phase. Table 30 lists the permitted sequences of services which may be used to negotiate the value for an individual parameter.

Sequences 1, 3 and 5 successfully negotiate a new value for a VTE-parameter. Sequences 2, 4 and 6 are unsuccessful attempts to negotiate a new value for a VTE-parameter; the value remains unchanged. Sequences 3 and 4 are "counter-offer" sequences. A counter-offer is not permitted after an INVITE. A counter-offer may only contain values for VTE-pa-

Table 30 - Valid MIN sequences

Sequence Number	MIN service issuable by alternate VT-users subject to dialogue constraints in 27.3.1 and 27.3.2 (time sequence is left to right)		
1	VT-NEG-OFFER	VT-NEG-ACCEPT	
2	VT-NEG-OFFER	VT-NEG-REJECT	
3	VT-NEG-OFFER	VT-NEG-OFFER	VT-NEG-ACCEPT
4	VT-NEG-OFFER	VT-NEG-OFFER	VT-NEG-REJECT
5	VT-NEG-INVITE	VT-NEG-OFFER	VT-NEG-ACCEPT
6	VT-NEG-INVITE	VT-NEG-OFFER	VT-NEG-REJECT

rameters which have been the subject of an earlier OFFER from the peer VT-user; however, the values which may be contained in the counter-offer are not constrained by those in the previous OFFER. No restrictions, other than those specified in this paragraph, apply to order, grouping or interleaving of the MIN service primitives in the MIN negotiation sequences.

All initiated MIN sequences must be completed before a VT-END-NEG may be initiated.

NOTES

1 Each MIN primitive may contain data for one or more VTE-parameters. The negotiation of the value for each VTE-parameter is logically independent of the negotiation of values for all other VTE-parameters. Thus negotiations of values for several VTE-parameters may proceed in parallel.

2 The consistency and completeness of the set of VTE-parameter values is checked when an attempt is made to leave MIN and use the draft-VTE as a new full-VTE (using VT-END-NEG), see 17.2.

3 Special profile arguments are negotiated by VT-START-NEG; they may not be used in the MIN services given in the sequences in table 30.

30.3.1 Specific information for S-mode

In S-mode, the WAVAR access-right is used to control the MIN dialogue. Only the owner of this access-right may issue any of the MIN services or issue a VT-END-NEG to terminate MIN.

NOTE – In S-mode, either VT-user may initiate a MIN sequence but only when in possession of the WAVAR access-right.

30.3.2 Specific information for A-mode

In A-mode, the VT-users are assigned roles in MIN as follows:

a) the VT-user which initiated the VT-START-NEG which caused entry into the Negotiation Active phase is the **MIN-initiator.**

b) the VT-user which accepted this VT-START-NEG is the **MIN-acceptor.**

In A-mode, only the *MIN-initiator* may start a MIN sequence, i.e., initiate the first of the MIN services as given in table 30. A MIN sequence, once started, is continued with the two VT-users alternately issuing their choice of a valid MIN service according to table 30 until a valid sequence is completed. There is no time constraint on when the services in the sequence must be issued.

Only the *MIN-initiator* may request termination of MIN by issuing a VT-END-NEG.

NOTES

1 There is always a unique MIN-initiator and MIN-acceptor assignment as the service provider resolves collisions between VT-START-NEG requests, see 27.5.

2 The MIN-acceptor may not initiate an offer but may initiate a counter-offer for any VTE-parameters which are the subject of a previous offer.

31 Data Transfer facility

This clause defines the VT service which supports the Data Transfer facility, see 7.4.

31.1 VT-DATA service

31.1.1 Purpose

To update objects in a manner depending on the implicit or assigned update priority of the objects and, for A-mode, indicate if echoing may take place following processing of these updates.

31.1.2 Structure

See table 17.

31.1.3 Service parameters

Table 31 specifies the service parameters for the VT-DATA service together with an indication of when each parameter is required, see clause 26.

Table 31 - VT-DATA service parameters

Parameter Name	Req.	Ind.		
VT-object-update	M	M		
VT-echo-now	O	O=		
VT-start-entry	C	C=		

31.1.3.1 VT-object-update is used to convey an update to a display object or a control object. A VT-DATA request or indication primitive contains a sequence of VT-object-update service parameters. Each VT-object-update service parameter is composed of two sub-parameters, **VT-object-ident** and **VT-object-data**, which identify the object to be updated and the data with which it is to be updated. A single VT-DATA request primitive may thus update one or more objects; however, these objects are required to be of the same update priority, see 20.1.4 and 24.5. Further, at most one control object with trigger characteristic may be included in the set of objects being updated and it must be the target of the final VT-object-update service parameter in the list.

VT-object-ident must correspond to the name of a display object or control object existing in the current-VTE. Device object names may not be included (these are not updated directly by VT-DATA).

VT-object-data is a sequence of zero or more operations selected from the updates available for the type of object identified by *VT-object-ident*. The permitted updates for display objects are defined in clause 19.

Where *VT-object-ident* identifies a control object, the VT-object-data service parameter value takes one of the following forms:

a) if *CO-structure*=1 (or defaults): a single value corresponding to the value type and range defined for the CO;

b) if *CO-structure* is an integer >1: zero or more update elements of the form

CO-element-id CO-element-value

where CO-element-value must correspond to the value type and range for the identified data element;

c) if *CO-structure* is "non-parametric": the possible updates are specified by the definition identified by *CO-type-identifier*.

When *VT-object-ident* identifies a RIO, the *VT-object-data* parameter value must correspond to the definition of RIO operations given in clause 22.

31.1.3.2 VT-echo-now is an optional service parameter used only in A-mode; it may only be included when the identified objects have "normal" update priority. If present, it indicates that echoing may take place following processing of the Service Data Unit (see 24.5 and clause B.7).

NOTE – The use of *VT-echo-now* is independent of trigger or delivery characteristics. However, it will usually be desirable to cause implicit or explicit delivery immediately after a VT-DATA with *VT-echo-now* present.

31.1.3.3 VT-start-entry is an optional boolean service parameter used only in A-mode operation when Fields functional unit is selected (and is useful only when structured data entry control objects exist). A "true" value of VT-start-entry indicates to the receiving VT-user that data entry, under the control of FEICO and FEPCO records, may commence on completion of the updates contained in the VT-DATA primitive; a "false" value, the default if absent, indicates that data entry should not start.

NOTE – A reason for VT-start-entry having the value "false" might be that further updates to the display object, FDCO, FEICO, etc., are needed before data entry can take place.

31.1.4 Usage and effects

The rules of 27.4 apply to this service.

The VT-user must satisfy the access-rules for all the objects being updated, and must not be waiting for receipt-acknowledgement, see clause 32 (but is not constrained by a request for acknowledgement from the peer VT-user).

The set of *VT-object-update* service parameters are processed in the order in which they occur in the request primitive. The implications of the various values of VTE-parameter *type-of-delivery-control* on the effects of one or more VT-DATA update packages are defined in clause 24. Clause 24 also defines the conceptual queues held by the service provider associated with update primitives of display and control objects, see 24.5.

In S-mode, if an update to a control object with trigger characteristic is made, implicit delivery is caused, see clause 24, and a VT-GIVE-TOKENS indication primitive is initiated as well as and logically following the VT-DATA indication and ownership of the WAVAR access-right is deemed to be passed (see also 33.1).

In A-mode, an update to a control object with trigger characteristic causes implicit delivery, see clause 24.

32 Delivery Control facility

This clause defines the VT services which support the Delivery Control facility, see 7.5.

The type of delivery control in effect is controlled by the value assigned by the VTE-parameter *type-of-delivery-control* (see clause 24).

32.1 VT-DELIVER service

32.1.1 Purpose

To force delivery of and to indicate a delivery point in a sequence of VT-DATA service initiations, and, optionally, request acknowledgement of receipt of these.

32.1.2 Structure

See table 17.

32.1.3 Service parameter

Table 32 specifies the service parameter for the VT-DELIVER service together with an indication of when the parameter is required, see clause 26.

Table 32 - VT-DELIVER service parameter

Parameter Name	Req.	Ind.		
VT-ack-request	O	O=		

VT-ack-request is used by a VT-user to request acknowledgement of receipt of updates by the peer VT-user. It takes one of the values "no-acknowledgement" or "acknowledgement". The default is "no-acknowledgement".

32.1.4 Usage and effects

The rules of 27.4 and 27.5 apply to this service.

The service may be initiated by either VT-user to ensure the delivery of pending updates to objects with "normal" update priority; it is not available if VTE-parameter *type-of-delivery-control* has value "no-delivery-control", see clause 24.

The VT-DELIVER request primitive causes the service provider to initiate any pending VT-DATA indication primitives for all update priorities as defined in clause 24, and marks a delivery point in the stream of updates for the benefit of the VT-user in receipt of these VT-DATA indication primitives.

The service provider records any request for acknowledgement of update receipt. When such acknowledgement has been requested, the VT-user receiving the VT-DELIVER indication primitive should initiate a VT-ACK-RECEIPT service at an appropriate time (not constrained by the operation of the VT service). After initiating a VT-DELIVER requesting acknowledgement, a VT-user shall not issue any updates to objects

subject to delivery control until the corresponding VT-ACK-RECEIPT has been received. The VT-user shall also not initiate VT-RELEASE, VT-SWITCH-PROFILE or VT-START-NEG in both A-mode and S-mode or VT-GIVE-TOKEN in S-mode.

In A-mode, a VT-user receiving a request for acknowledgement is not constrained from issuing VT-DATA requests or VT-DELIVER requests.

If a VT-RELEASE indication primitive has been received and a VT-RELEASE response is pending, VT-DELIVER service can be initiated but shall not have *VT-ack-request* service parameter set to value "acknowledgement". See 27.5 for further information on collisions involving the VT-DELIVER service.

32.2 VT-ACK-RECEIPT service

32.2.1 Purpose

To acknowledge receipt of a delivery point marker indicated by the occurrence of a VT-DELIVER service.

32.2.2 Structure

See table 17.

32.2.3 Service parameters

None

32.2.4 Usage and effects

The service may only be initiated by a VT-user which has received a VT-DELIVER indication primitive requesting its use. The VT-user receiving the VT-ACK-RECEIPT indication primitive may use it to synchronise the activities of the two VT-users (the activities being synchronised are beyond the scope of the VT service).

The VT-user receiving the VT-ACK-RECEIPT indication may now issue any of the service requests restricted by 32.1.4. VT-ACK-RECEIPT does not force delivery. It is sequenced with respect to VT-SWITCH-PROFILE, VT-START-NEG and VT-RELEASE; however, it is not necessarily sequenced with respect to updates to any display or control object.

33 Dialogue Management facility

This clause defines the VT services which support the Dialogue Management facility, see 7.6.

33.1 VT-GIVE-TOKENS service

33.1.1 Purpose

To pass ownership of the WAVAR access-right, currently owned by the initiating VT-user, to the peer VT-user.

33.1.2 Structure

See table 17.

33.1.3 Service parameters

None.

33.1.4 Usage and effects

The rules of 27.4 apply to this service. This service is only valid in S-mode and can be initiated only by the VT-user which currently owns the WAVAR access-right. The VT service provider records the change in ownership of the WAVAR access-right. Delivery of outstanding updates is implied.

NOTE – When an update is made to a "trigger" control object, the single VT-DATA request primitive results in two indication primitives, a VT-DATA indication primitive for the update and a VT-GIVE-TOKENS indication for the WAVAR access-right being passed as a result of the trigger.

33.2 VT-REQUEST-TOKENS service

33.2.1 Purpose

To request a transfer of ownership of the WAVAR access-right when not currently owned by the initiating VT-user.

33.2.2 Structure

See table 17.

33.2.3 Service parameters

None.

33.2.4 Usage and effects

The rules of 27.4 apply to this service.

This service is only valid in S-mode and can be initiated only by the VT-user which currently does not own the WAVAR access-right. If the peer VT-user has already issued a VT-GIVE-TOKENS request primitive, the VT-REQUEST-TOKENS indication primitive is not delivered to that VT-user. There is no recorded effect on the VT service provider due to the execution of this service.

34 Destructive Interrupt facility

This clause defines the VT service which supports the Destructive Interrupt (Break) facility, see 7.7.

34.1 VT-BREAK service

34.1.1 Purpose

To interrupt the activities of two VT-users and discard all previously initiated object updates which have not been processed.

34.1.2 Structure

See table 17.